Iniciación a los Microprocesadores

E.A. PARR



Monografías CEAC de Informática

Iniciación a los microprocesadores

MONOGRAFIAS CEAC DE INFORMATICA

Iniciación a los microprocesadores

E.A. PARR



Perú, 164 - Barcelona 20 - España

Este libro está basado en una publicación titulada *Introduction to Microprocessors and Computing*, publicado por el autor en edición limitada.

El autor y los editores dan las gracias a INTEL CORPORATION por permitirles reproducir ciertos códigos y descripciones del juego de instrucciones que figura en el manual MCs 8080/85.

Aunque la preparación de este libro se ha hecho con todo cuidado, ni los editores ni el autor serán responsables en modo alguno de ningún error que pudiera contener.

Traducción autorizada de la obra:
A MICROPROCESOR PRIMER
Editado en lengua inglesa por
BERNARD BABANI LTD.
© 1980 BERNARD BABANI LTD.
ISBN: 0-900162-92-9

© EDICIONES CEAC, S. A.
Perú, 164 - Barcelona 20 (España)

2.ª edición: Abril 1983

ISBN 84-329-6601-0

Depósito Legal: B. 17719 - 1983

Impreso por
GERSA, Industria Gráfica
Tambor del Bruc, 6
Sant Joan Despí (Barcelona)

Printed in Spain
Impreso en España

INDICE

1. INTRODUCCION

El principiante en el campo de la electrónica suele sentirse abrumado cuando se enfrenta por primera vez con un artículo sobre microprocesadores. Generalmente, los problemas que encuentra se agrupan en tres categorías. En primer lugar, tenemos una abundantísima terminología nueva, y es una triste realidad que la mayoría de los artículos y hojas de datos dan por sentado que el lector sabe lo que significan punteros de pila, unidades UART de direccionamiento indexado y todo el resto de la jerga. En segundo lugar, el lector tiene la comprensible sensación de estar recibiendo un chorro de información y en estas condiciones puede comprender pequeñas dosis de tecnología, pero no situarse a distancia y apreciar el cuadro en su conjunto ("los árboles no dejan ver el bosque"). Por último, gran parte de la lógica de microprocesadores parece terriblemente arbitraria; por ejemplo, por qué 3E representa la dirección extendida del registro de carga A en un micro Z80, y este código se limita al registro A y no está disponible en B, C, D, E.

En un intento de ofrecer un planteamiento sencillo del cálculo automático, este libro comenzará con el diseño de un ordenador simple al que llamaremos Microprocesador Digital Integrado Mark 1, o DIM-1 para abreviar (del inglés, *D*igital *I*ntegrated *M*icroprocessor). Será una máquina totalmente básica, con una tecnología similar a la empleada en los ordenadores a comienzos

de los años sesenta. El DIM-1 está dotado de un conjunto de instrucciones muy limitado, pero, lógico. Debido a su sencillez y a su estructura lógica, el lenguaje resulta fácil de aprender y esperamos que el lector logre evitar el interminable repaso de las hojas de instrucciones que suele ser tarea habitual cuando se estudia por primera vez un microprocesador.

Por motivos puramente de interés, el DIM-1 está basado en una máquina real y el lector que haya tenido ocasión de conocer el ordenador eléctrico inglés llamado KDN2 observará un parecido más que casual. ¡Qué tiempos aquellos en que un ordenador de 4K pesaba 2 toneladas, necesitaba 5KW de potencia y ocupaba 2 metros cuadrados de espacio!

Con la explicación del DIM-1, se apreciarán naturalmente algunas (¡muchas!) de las carencias de la máquina y también que éstas pueden ser superadas realizando cambios y adiciones en el conjunto de instrucciones. En este sentido desarrollaremos conceptos tales como el direccionamiento relativo, registros de índice y otros similares, con la esperanza de que sean vistos como avances lógicos y no como arbitrariedades que han de ser aceptadas aunque no se las comprenda.

En un libro de este tipo es necesario suponer que el lector sabe contar en binario, y que tiene una vaga idea de lo que son los número hexadecimales y octales. Se da por sentado así mismo que tiene conocimientos básicos sobre circuitos lógicos. En el apéndice 1 se hace una breve introducción al sistema binario.

2. ARQUITECTURA DEL ORDENADOR

Antes de proceder al diseño del DIM-1, veamos qué es lo que se necesita en un ordenador. Para facilitar la explicación, supongamos que hemos de realizar una serie de cálculos repetitivos con la colaboración de un ayudante que, aunque torpe, sabe de letras y números. Como trabajo típico a realizar, podríamos estar calculando y enviando recibos de la luz.

Para empezar, sentaremos a nuestro ayudante en un escritorio, le proporcionaremos una calculadora electrónica, y hojas de papel para anotar cálculos parciales. A continuación le daremos un archivador con los datos que ha de utilizar almacenados de manera lógica y accesible. Finalmente, y lo que es más importante, le facilitaremos un conjunto de instrucciones que ha de seguir. Dependiendo de su capacidad para tomar decisiones, estas instrucciones pueden ser muy generales (multiplique las unidades consumidas por 2,15 ptas., para determinar el importe del recibo) o muy específicos (pulse la tecla de borrado en la calculadora, cargue el número de la columna 1 en la calculadora, pulse la tecla X, pulse 2,15, pulse la tecla =, anote el resultado en la columna 7 del recibo).

Entre este proceso y la manera de trabajar de un ordenador existe una gran semejanza. Simplificando las cosas al máximo, un ordenador puede ser representado mediante la figura 1.

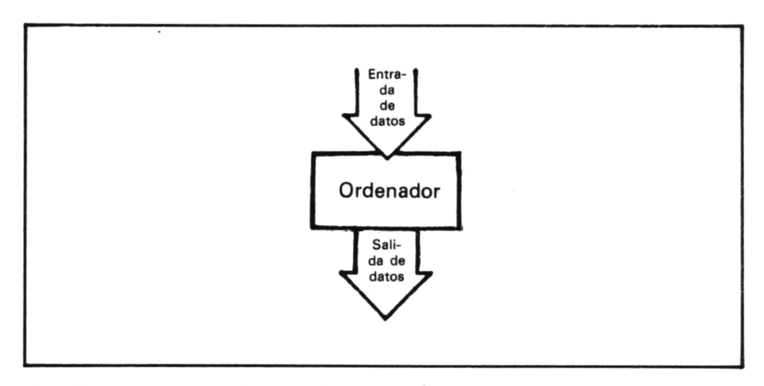


Fig. 1 Diagrama de bloques del ordenador.

La información o datos (como se le quiera llamar) se carga en el ordenador, se procesa, y sale nueva información. Pero lo que verdaderamente nos interesa es lo que ocurre dentro del bloque que representa al ordenador.

En primer lugar, necesitamos un conjunto de instrucciones que hemos de seguir. Sin definir por ahora cómo hemos de escribirlas, imaginemos que las instrucciones se encuentran en las casillas con las etiquetas instrucción 1, instrucción 2, instrucción 3, etc., y que el ordenador comenzará con la instrucción 1, avanzando paulatinamente paso a paso. Nosotros podemos identificar algunas instrucciones que nos serán útiles. STOP es la primera de ellas, y resulta suficientemente clara si se piensa en su significado.

La segunda no es tan fácil de definir, pero no resulta difícil comprender que un ordenador que solamente avance de instrucción en instrucción tendría un uso bastante limitado. Sería muy útil poder decir: "Si el último resultado fue X, continúe con la siguiente instrucción; pero si el resultado no fue X, no continúe, vaya a la instrucción número N y prosiga a partir de ese punto." Esto nos permitiría escribir programas flexibles. Volviendo nuevamente a nuestro ayudante y al recibo de la luz, podríamos comprobar si hay partidas pendientes para cada consumidor y, si el resultado fuese afirmativo, enviarle la correspondiente notifica-

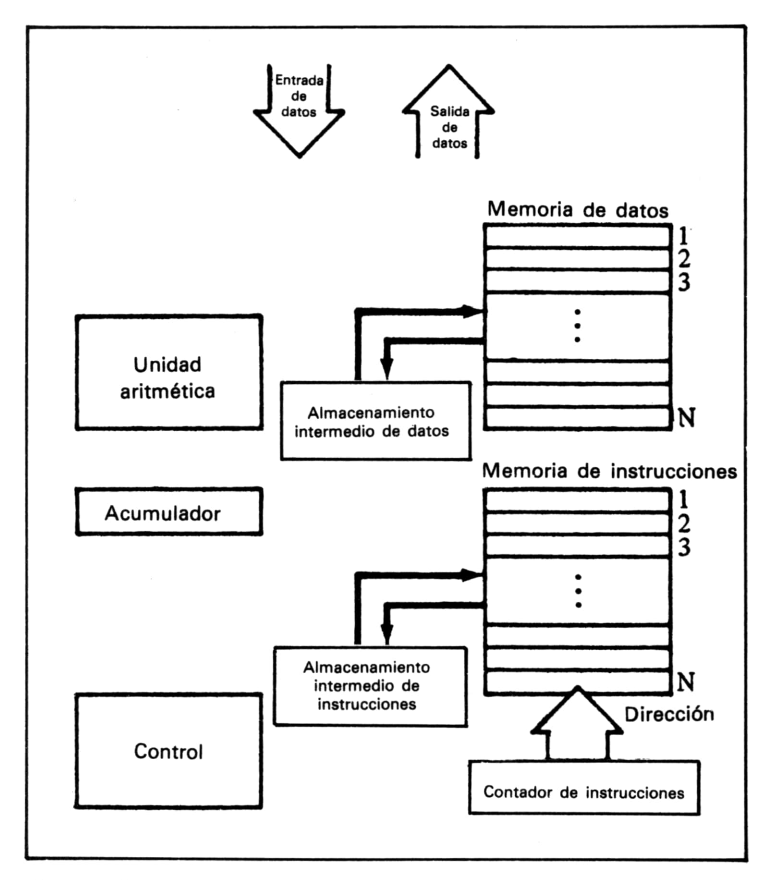


Fig. 2 Diagrama de bloques ampliado.

ción en los términos adecuados. Finalmente necesitaremos las instrucciones aritméticas de suma y resta.

De esta manera, la sustitución de nuestro ayudante por un equipo electrónico puede resumirse como se ilustra en la figura 2. En la parte superior del diagrama tenemos las palabras entrada de datos y salida de datos, que representan nuestra comunicación con la máquina. En el centro del diagrama tenemos la unidad

aritmética, que equivale a la calculadora. A continuación se encuentra un bloque llamado acumulador consistente en un conjunto de controles biestables capaces de contener un número en binario. En el DIM-1 hay 16 controles biestables, de modo que puede contener cualquier número entre 0000000000000000 (cero) y 1111111111111111 (65535 en decimal). El acumulador es el equivalente de la hoja de papel y se usa para almacenar resultados intermedios.

A la derecha del acumulador se encuentra la memoria de datos que por su estructura suele ser comparada a una hilera de casillas de palomas, cada una con una dirección única. Cada casilla puede alojar un número. En el DIM-1 hay 4096 casillas que pueden contener otros tantos números binarios de 16 bits, dentro de los límites indicados en el párrafo anterior. El lector podrá preguntarse por qué un número tan extraño como el 4096. La respuesta es que cuando tengamos que dirigirnos a la memoria lo haremos en binario, y un número binario de doce bits nos permite el acceso a todas las posiciones comprendidas entre 000000000000 y 111111111111. El último número es 4095 en el sistema decimal, y por tanto hay 4096 posiciones a las que podemos dirigirnos con un número de doce bits. Esto es lo que se conoce como memoria de 4K (K indica mil, un tanto libremente). Generalmente, las memorias de los ordenadores están formadas por bloques de 4K, de modo que pueden encontrarse memorias desde 4K hasta 64K (actualmente 65536). Los equipos microprocesadores se venden a menudo con memorias de 1K, lo que representa 1024 posiciones.

Asociado a la memoria está el almacenamiento intermedio de datos. Este es un registro de 16 bits, similar al acumulador, cuya función consiste en almacenar los datos sacados de la memoria mientras son utilizados. Del mismo modo, los datos que han de ser colocados en la memoria pasarán antes por el almacenamiento intermedio de la memoria.

A continuación, en una memoria de 4K idéntica a la de datos, tenemos las instrucciones. Más adelante veremos que esto es una complicación innecesaria y que los datos e instrucciones se almacenan en la misma memoria. No obstante, por el momento consi-

deraremos que ambas están separadas. La memoria de instrucciones es idéntica a la de datos, que, como hemos visto, está formada por una hilera de casillas, cada una capaz de almacenar un número de 16 bits. Estos números representan instrucciones para la máquina de manera que, por ejemplo, el número 1111000000000000 significa Stop. El acceso a las posiciones de esta memoria también se realiza mediante una dirección única.

Asociados con la memoria de instrucciones aparecen el contador de instrucciones y el almacenamiento intermedio de instrucciones. El contador de instrucciones es un registro que se usa para contener la dirección de la instrucción en curso. La memoria de instrucciones tiene 4K posiciones y, por consiguiente, son necesarios 12 bits para acceder a ella. El contador de instrucciones es por tanto, un registro de 12 bits.

En un programa normal avanzamos de instrucción en instrucción, de manera secuencial, a menos que encontremos la instrucción especial: "si ocurre X haga esto, sino haga lo otro" que mencionábamos anteriormente. Después de haber ejecutado una instrucción, el contador ha de ser incrementado para que nos indique la dirección de la siguiente instrucción. Por eso sería útil convertir este registro en un simple contador binario, ya que cada vez que ejecutásemos una instrucción sólo tendríamos que enviar un impulso al contador para obtener la dirección de la siguiente. La instrucción condicional requiere la incorporación de otros medios en el contador de instrucciones, pero sobre este tema volveremos más adelante.

La memoria de instrucciones cuenta también con un almacenamiento intermedio de instrucciones que realiza el mismo cometido que el almacenamiento de la memoria de datos. Al iniciarse el ciclo de ejecución de una instrucción, el contador se dirige a la memoria de instrucciones, y el número que se halle en la posición buscada es colocado en el almacenamiento intermedio de instrucciones. Este número representa la instrucción que ha de ser ejecutada.

Finalmente tenemos el control. Este enlaza todas las partes del ordenador y se encarga de que cada una de ellas lleve a cabo

su cometido en el momento preciso, por lo que en cierta manera puede ser considerado como el "conductor". Además, el control tiene la misión de comprobar qué número se encuentra en el almacenamiento intermedio de instrucciones y de decidir a qué instrucción representa.

Para comprender cómo se coordinan todas estas partes, veamos por donde se encamina el ordenador cuando ejecuta una instrucción. La instrucción en cuestión se halla en la posición 3220 y el número allí contenido representa: "Sumar el número que está en la posición 4057 al número del acumulador y colocar el resultado en el acumulador." El número que pueda representar esta particular instrucción no nos preocupa por el momento; nos basta con saber que el control lo reconocerá.

Comenzaremos a seguir la máquina al final de la instrucción anterior (3219). El contador indicará la instrucción 3219. Cuando esta finaliza, el control envía un impulso al contador de instrucciones incrementándolo a 3220. En ese momento el control ordena a la memoria de instrucciones que lea, y el número que se encuentra en la posición de memoria 3220 es transferido al almacenamiento intermedio de instrucciones. El control examina este número y decide qué significa: "Sumar el número que está en la posición 4057 al número que se encuentra en el acumulador, y colocar el resultado en el acumulador."

Ahora el control se dirige ("direccional") la memoria de datos con la dirección 4057 y ordena su lectura. El número de la posición 4057 está ahora en el almacenamiento intermedio de la memoria, y el control exige a la unidad aritmética que sume los números contenidos en el acumulador y en el almacenamiento intermedio de la memoria. Una vez realizada esta operación, el control toma el resultado de la unidad aritmética y lo coloca en el acumulador.

Por último, el control envía un impulso al contador de instrucciones para incrementarlo a 3221, que es el número correspondiente a la siguiente instrucción. Esta acción completa el ciclo de ejecución de la instrucción.

Supongamos que la instrucción 3221 es: "Poner el número que se halla en el acumulador, en la posición de memoria 1750." Sigamos los pasos de la máquina para ver cómo la ejecuta.

Ya hemos incrementado el contador de instrucciones y contiene el número 3221. El control se dirige a la memoria de instrucciones y le ordena que lea el número contenido en la posición 3221 y lo transfiera al almacenamiento intermedio de instrucciones, donde el control lo identifica como: "Poner el número que se halla en el acumulador, en la posición de memoria 1750."

El control toma el número del acumulador y lo coloca en el almacenamiento intermedio de la memoria de datos. Después envía la dirección 1750 a la memoria de datos y le ordena que escriba en la posición 1750 los datos que están en el almacenamiento intermedio de la memoria. El control incrementa a 3222 el contador de instrucciones mediante el correspondiente impulso, y con ello se completa el ciclo de ejecución.

Como puede verse, cada ciclo se divide en dos partes distintas. En la primera el control se dirige a la memoria de instrucciones, transfiere la instrucción al almacenamiento intermedio, y "descifra" su significado; esto es, decide lo que representa el número que se encuentra en el almacenamiento intermedio.

En la segunda parte, el control ejecuta la instrucción. Normalmente esta operación implica el uso de la memoria de datos, ya que casi siempre se manejan datos de la memoria y datos del acumulador.

Esta segunda parte es común para todos los ordenadores y se le ha llamado "latido" del ordenador. A estas dos partes se les llama a menudo ciclo de instrucción y ciclo de datos.

El control en ningún momento accede simultáneamente a la memoria de datos y a la de instrucciones. Por consiguiente, en la figura 3 hemos simplificado el equipo utilizando sólo una memoria para datos e instrucciones. Esto es perfectamente factible dado que tanto los datos como las isntrucciones se almacenan en la memoria en forma de números de 16 bits y la memoria no conoce el significado de su contenido.

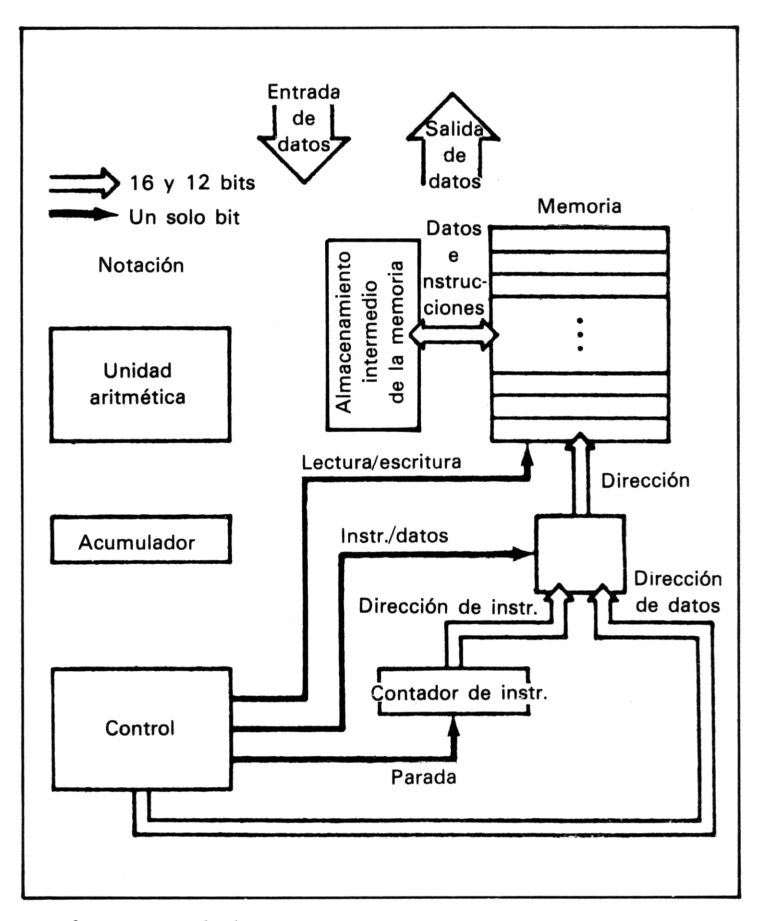


Fig. 3 Memoria de datos e instrucciones.

El uso de la misma memoria para almacenar datos e instrucciones tiene una ventaja muy importante que posiblemente no se haga evidente de inmediato. No es sólo la propia memoria la que ignora si el número en una posición determinada representa una isntrucción o representa datos; el ordenador, en su conjunto, también lo ignora hasta que se produce el acceso a la posición

mediante el ciclo de instrucción o el ciclo de datos. Esto significa que una instrucción puede ser tratada como datos y manejada por otras instrucciones.

De buenas a primeras esto puede parecer un poco alarmante, y para algunos la idea de que un ordenador modifique sus propias instrucciones puede resultarles totalmente siniestra. Pero como veremos más adelante al construir nuestro conjunto de instrucciones, el uso de este método no es cosa de otro mundo.

3. REPRESENTACION DE NUMEROS Y FORMATO DE INSTRUCCIONES

Antes de enumerar el conjunto de instrucciones del DIM-1, pensemos un poco sobre el aspecto que tendrá la instrucción. Dado que cada posición, registro, etc., tiene una longitud de 16 dígitos (o bits) binarios, la instrucción será un número de 16 bits como el siguiente:

0001100000101111

A primera vista esto poco significa, y sería deseable disponer de un medio de representación más adecuado que nos permitiese reconocer la instrucción. Obviamente, uno de ellos podría ser la conversión de la representación binaria a su equivalente decimal. Esto nos facilitaría un número que podríamos recordar pero que, como veremos más adelante, no nos diría realmente mucho sobre la instrucción. La solución ideal a nuestras necesidades ha de ser un método de representación que, nada más ver la instrucción, nos permite decir: "Ah sí, eso quiere decir que hay que acceder a la posición de memoria 4057 y transferir su contenido al acumulador."

La mayoría de las instrucciones utilizan posiciones de memoria (sumar, restar, llevar al acumulador, transferir a la memoria) y de ello se deduce que en alguna parte de la instrucción hemos de poder especificar una dirección de memoria. El DIM-1 tiene

una memoria de 4K, y, como hemos mencionado anteriormente, con 12 bits podemos acceder a cualquiera de sus posiciones. Por consiguiente, dividamos la instrucción en dos partes; los 12 bits menos significativos definirán la dirección, y los cuatro primeros la operación a realizar:

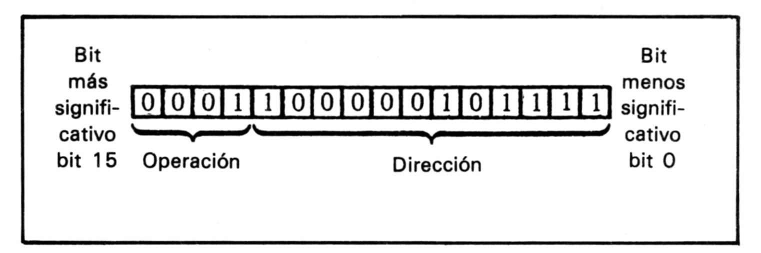


Fig. 4.

Con cuatro bits podemos definir 16 operaciones diferentes (0000-1111 en binario), y pronto decidiremos cuales van a ser. Por ahora lo que necesitamos es algún medio rápido y simple para identificar una instrucción; un medio con el que podamos distinguir las partes de operación y dirección. Por ésta razón, no nos sirve la representación decimal.

De hecho, hay dos métodos comunes de identificación que consisten en agrupar los dígitos binarios.

El primer método lo agrupa en bloques de tres, de la siguiente manera:

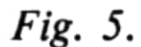
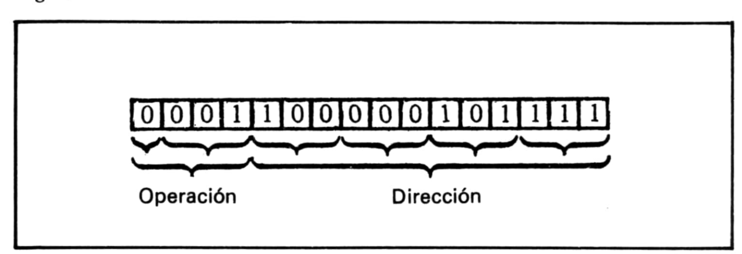


Fig. 5.



Cada bloque de tres dígitos puede representar un número entre 0 y 7 (000-111). La instrucción se representa mediante 5 números entre 0 y 7, más el número de orden superior que puede ser un 0 o un 1. La parte de dirección es un número comprendido entre 0000 y 7777, y la parte de operación uno entre 00 y 17.

Tomamos la instrucción de la figura 5.

0001100000101111

La fraccionamos

0 001 100 000 101 111

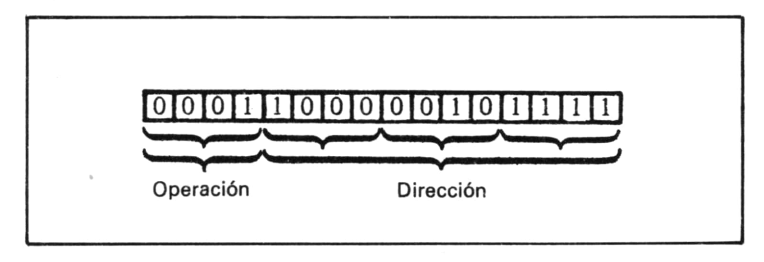
Escribimos debajo los números a que equivale

0 1 4 0 5 7

y podemos decir que esta es una operación '01' que utiliza la posición de memoria 4057. Hemos de tener presente que este número NO es el 4057 del sistema decimal, sino un número con base 8, llamado octal. En un número octal nunca puede existir un 8 ó un 9.

El segundo método agrupa los bits en bloques de cuatro, de la siguiente manera:

Fig. 6.



Esto nos da cuatro números de cuatro bits con los que podemos representar otros tantos números entre 0 y 15 (0000-1111), aunque para ello necesitemos ampliar el sistema decimal como se muestra a continuación:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Las letras utilizadas en la ampliación tienen la siguiente equivalencia en binario:

8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Para representar una instrucción de esta manera comenzamos por dividirla en bloques de cuatro bits:

	0001100000101111			
se convierte en	0001	1000	0010	1111
sustituyendo a	1	8	2	F

El número uno representa la instrucción, y los tres últimos la dirección. Por consiguiente, ya podemos mirar a esta instrucción y decir: "Sí, ésta es la instrucción '1', en la posición de memoria 82F."

Este método se conoce con el nombre de representación hexadecimal. La representación de una instrucción en octal como (014057) o en hexadecimal (182F), depende generalmente de preferencias personales y de las hojas de datos del fabricante del ordenador. Para el DIM-1 utilizaremos el sistema octal porque nos permite dar un formato fácilmente reconocible a las 16 instrucciones y porque el aspecto de las instrucciones codificadas en

octal resulta menos impresionante para el principiante que las codificadas en hexadecimal.

Cualquiera que sea el método elegido por el fabricante, el ordenador siempre tendrá la "lógica" que permite cargar programas en octal o en hexadecimal, y la lógica interna de la máquina los convertirá a binario (actualmente, una tarea fácil). El programador no tiene que descender a la complejidad que supone el trabajo con bits (¡al menos no tiene que hacerlo a menudo!) y piensa principalmente en octal o hexadecimal. Por otra parte, el ordenador piensa siempre en binario, independientemente del sistema de representación que use el programador.

Veamos el aspecto de una instrucción en el DIM-1. Es una representación octal de un número de 16 bits. Una instrucción típica podría ser la siguiente:

073220

La instrucción se divide en dos partes. Los cuatro últimos números representan la posición de memoria en octal, y nos permiten decir de inmediato que la instrucción utiliza la posición 3220. Los dos primeros números representan la operación que se va a realizar.

Llegado este momento probablemente lo mejor será decidir el modo de asignar las dieciséis instrucciones que tenemos.

En primer lugar necesitamos desplazar datos desde la memoria al acumulador y desde éste a la memoria. A estas dos operaciones les llamaremos respectivamente "Acceso" y "Almacenamiento".

A continuación desearíamos realizar, por lo menos, operaciones de aritmética básica, y nos vendrían muy bien las cuatro funciones de una calculadora (suma, resta, multiplicación y división). De hecho, son pocos los ordenadores que tienen las instrucciones de multiplicar y dividir como instrucciones normales. Ello se debe a que con las de suma, resta y otras que veremos más adelante, se pueden escribir fácilmente las rutinas de multi-

plicar y dividir de un programa. Por tanto, el DIM-1 sólo contará con dos instrucciones aritméticas; Suma y Resta.

La siguiente instrucción que vamos a necesitar será de tipo lógico. Muchas veces ocurre que la información a procesar no consiste en números ni en instrucciones, sino en una representación de datos. Por ejemplo, podríamos representar el estado de 16 interruptores limitadores en una nave industrial, o asignar cada bit a una característica específica en un sistema de proceso de datos, como se muestra a continuación:

Bit 0	1 Varón,	0 Mujer
Bit 1	1 Mayor de 65,	0 Menor de 65
Bit 2	1 Menor de 16	0 Mayor de 16

y así sucesivamente.

Existen dos funciones lógicas, cuya utilidad ha sido demostrada por la experiencia. La primera de ellas es la función Y (AND), que permite almacenar una "MASCARA" en la memoria para extraer parte de los datos del acumulador. Supongamos que hemos leído el estado de 16 interruptores limitadores en algún dispositivo de una nave industrial. Los interruptores limitadores relacionados, por ejemplo, con un generador se encuentran en los bits 3, 2, 1, 0. Si en la posición 7077 almacenamos la máscara 000017 (en octal) y después ejecutamos AND7077, los datos del generador quedarán en el acumulador y los bits 15-4 contendrán todos cero.

A la segunda función lógica le llamaremos "comparación", aunque su nombre propio es O exclusiva (OR exclusiva en la jerga). Supongamos que tenemos en la lista de datos en forma de palabra de 16 bits, correspondientes a varias personas. Cada bit representará una característica que puede ser descrita por el método Sí/No. Para ello hemos de realizar frecuentes búsquedas de los datos (p.ej.: cuántos varones, con vehículo propio, tienen una edad comprendida entre 30 y 40 años y leen la revista Selecciones). La función "comparación" nos permite utilizar una máscara y comprobar. En este caso, a diferencia de la función, además

de la coincidencia de los "0" binarios también nos interesa la de los "1". Si la función de comparación coloca un "0" en el acumulador cuando éste y la MASCARA coinciden en un determinado bit, y un "1" cuando no coinciden, el contenido del acumulador será cero para las coincidencias y cualquier otro número para la discordancias. La instrucción de salto condicional que anteriormente considerábamos necesaria, en realidad funciona basándose en el contenido cero de un acumulador que (como veremos más adelante) permite escribir el programa para ejecutar la acción que proceda.

Una definición más rigurosa de las instrucciones lógicas se dará en el resumen de instrucciones que sigue a este texto descriptivo.

El cuarto tipo de instrucción opera únicamente en el acumulador, y nos permite mover los datos en él contenidos a izquierda y derecha. A esta operación se le llama desplazamiento. La instrucción de desplazamiento es la única que NO contiene una dirección en los 12 bits menos significativos. De los 12 bits sólo se usan cuatro para indicar al control cuántas posiciones ha de desplazarse el acumulador. Dado que el acumulador es un registro de 16 bits, los desplazamientos de más de 16 posiciones carecen de sentido. Las instrucciones de desplazamiento son dos; desplazamiento a la izquierda y desplazamiento a la derecha.

Las funciones de desplazamiento, aparte de permitir la manipulación de datos en el acumulador, constituyen la base en los programas para multiplicar y dividir. Desplazando el acumulador en una posición a la izquierda, se multiplica por dos, desplazándolo dos posiciones se multiplica por cuatro, y así sucesivamente. Los desplazamientos a la derecha afectan a la división de manera similar.

El quinto tipo de instrucción lo constituye el salto condicional que mencionábamos antes. Esta instrucción permite a un programa comportamientos diferentes, dependiendo del estado del acumulador.

Antes de pasar a describir en detalle estas dos operaciones hablaremos un poco de números negativos y positivos, y de su representación en binario. Para simplificar las cosas, supongamos que tenemos una palabra de seis bits; el número decimal seis, se representaría como,

	000110
y el nueve como	001001
Si restamos nueve de seis,	111101
obtendremos que significa −3.	
Para comprobar la operación,	000011
le sumamos tres y el resultado será	(1)000000

Dado que el registro sólo tiene seis bits de longitud, el "1" binario de la izquierda se pierde, y el resultado cero muestra que 111101 es, en realidad, −3.

Métodos similares se aplican con registros de cualquier tamaño. Como norma, el dígito más significativo se usa como bit de signo, y es un "1" para un número negativo y un "0" para un número positivo. La regla para generar un número negativo es simple. Escribimos el número en binario, p. ej.

	0000000000001010	(diez en decimal)
lo complementaremos	1111111111110101	(es decir, sustituimos los '1' por '0', y viceversa)
y añadimos 1	1111111111110110	

Este resultado es el número negativo del binario original, y para eliminar toda duda podemos sumarlo al número original y comprobar si el resultado es cero.

Recordamos que, según nuestra definición, el cero es un número positivo. Recordemos, así mismo, que la representación octal de un número de 16 bits nos permite saber de inmediato si un número es positivo o negativo, gracias a que el bit más signifi-

cativo puede ser '0' ó '1'. Por ejemplo, 032756 es positivo y 170677 es negativo.

Volviendo a los saltos condicionales, hemos de contar con dos instrucciones. La primera comprueba si el acumulador es positivo o negativo. Si es positivo el programa salta a la instrucción indicada en la parte de dirección, si es negativo, continúa en secuencia con la siguiente instrucción.

El segundo salto condicional comprueba si el acumulador es cero; si no es cero, el programa salta a la instrucción indicada en la parte de dirección, y si es cero, continúa en secuencia con la siguiente instrucción.

Supongamos que estamos controlando un proceso industrial y que deseamos hacer sonar una alarma cuando se cierre un conmutador limitador hidráulico que controle el nivel mínimo de aceite. Daríamos entrada a un bloque de 16 conmutadores, de los cuales uno sería el indicador del nivel mínimo de aceite (Las instrucciones de Entrada y Salida se describen más adelante). Digamos que el conmutador en cuestión está en el bit 4 y que hemos almacenado la máscara 000020, que en binario equivale a 0000000000010000. Relacionamos la máscara con el acumulador mediante la función Y (AND), saltando a continuación a la parte de alarma del programa (el acumulador no es cero). Si el conmutador no se cierra, el acumulador será cero, y por tanto en vez de saltar, pasaremos simplemente a la siguiente instrucción del programa.

Cuando los programadores comenzaron a escribir programas, pronto se percataron de que una operación muy frecuente consistía en incrementar y disminuir el contenido de posiciones de memoria para contabilizar el número de veces que se realizaba una operación determinada.

Así pues a la sexta instrucción le llamaremos aritmética unitaria. Esta nos permite incrementar o disminuir el contenido de una posición de memoria específica y, por añadidura, dejar una copia del resultado en el acumulador.

Las dos instrucciones que necesitamos en séptimo lugar son simples. La primera ocasiona un salto incondicional; el programa salta simplemente a la dirección especificada y continúa a partir de ella.

La segunda produce un salto incondicional seguido de una parada. El programa salta a la instrucción específica, pero se detiene antes de ejecutarla. Para continuar el operador tienen que pulsar el botón "go", lo que hace posible una pequeña participación humana en el procedimiento.

Finalmente, tenemos las de entrada y salida. Estas son instrucciones bastantes detalladas, y en el apartado de especificación de instrucciones se da una descripción completa al respecto. Por el momento diremos que a los dispositivos de Entrada/Salida (I/O (Input Output) se les asignan direcciones (p. ej., teletipo n.º 3, contactor de motor 257, lectora de cinta 7) de manera que la parte de dirección de una instrucción especificará el dispositivo, y no una posición de memoria. La función de estas instrucciones consiste en transferir datos desde el dispositivo de entrada al acumulador y desde éste al dispositivo de salida.

En total hemos especificado dieciséis instrucciones, que se resumen a continuación:

Búsqueda	Almacenamiento	(Movimiento)
Sumar	Restar	(Aritmética)
Y (AND)	Nev (no equivalente)	(Lógica)
Desplazar a la izquierda	Desplazar a la derecha	(Desplazamiento)
Saltar si es positivo	Saltar si no es cero	(Condicional)
Incrementar	Disminuir ("decrementar")	(Aritmética unitaria)
Saltar	Saltar y parar	(Salto)
Entrada	Salida	(I/O)

Como puede verse, estas instrucciones, por su naturaleza, se clasifican en los ocho grupos indicados a la derecha. Sería lógico utilizar los bits 12, 13 y 14 para definir el grupo (p. ej., movimien-

to), y el bit 15 como señalizador para indicar cuál de las dos instrucciones estamos usando.

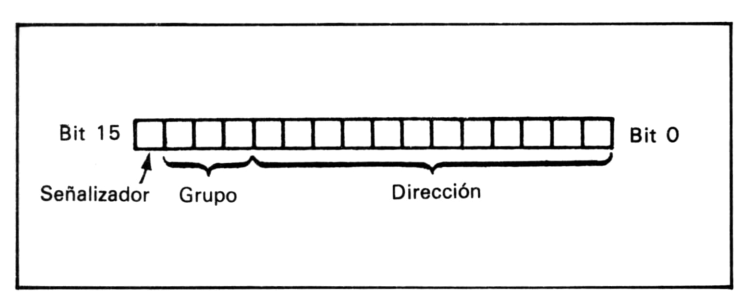


Fig. 7.

Por ejemplo, tanto "sumar" como "restar" tendrán un '2' en la parte del grupo "sumar" tendrá un '0' en el bit señalizador, mientras que "restar" tendrá un '1'.

Por consiguiente, 023220	significa sumar el contenido de la posición 3220 al acumulador
y 123220	significa restar el contenido de la posición 3220 del acumulador.

Este es un formato lógico y fácil de reconocer del que, desafortunadamente, carecen los conjuntos de instrucciones utilizados en microprocesadores. Llegado el momento de ampliar el conjunto de instrucciones del DIM-1; comprobaremos que el formato lógico comenzará a volverse un poco confuso.

A continuación se define el conjunto de instrucciones de manera tanto legalista, y se incluyen ejemplos de las instrucciones utilizadas. En cierto modo, se trata de repasar más detalladamente lo que acabamos de ver. Esta definición de instrucciones es similar a la que el lector encontrará en los manuales técnicos referentes a microprocesadores

4. DEFINICION DEL CONJUNTO DE INTRUCCIONES DEL DIM-1

Una instrucción del DIM-1 tiene el formato de una palabra de 16 bits, y se representa mediante su equivalente en octal. La palabra se divide en tres campos:

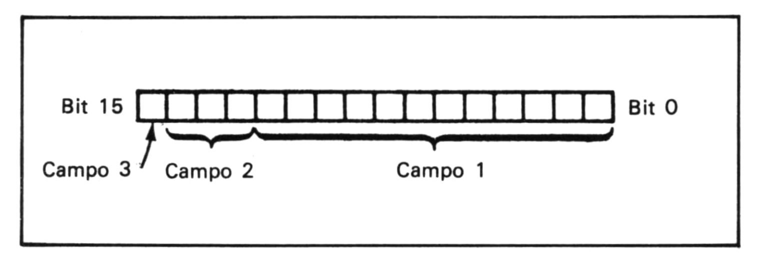


Fig. 8.

El campo 1 está formado por los bits del 0 al 11, y se llama campo de dirección.
El campo 2 está formado por los bits del 12 al 14, y se llama campo de operación.
El campo 3 lo constituye el bit 15, y se llama bit señalizador.

En el DIM-1 existen ocho operaciones cada una con dos modos operativos especificados por el bit señalizador. Estas operaciones se describen a continuación.

Operación 1. Movimiento

Esta operación traslada datos entre el acumulador y la memoria.

01 Búsqueda (p. ej. 017322)

El contenido de la posición de memoria especificada se coloca en el acumulador. El contenido previo del acumulador se pierde y el de la posición de memoria permanece inalterado.

11 Almacenamiento (p. ej. 113220)

El contenido del acumulador se almacena en la posición de memoria especificada. El contenido previo de la posición de memoria se pierde y el del acumulador permanece invariable.

Operación 2. Aritmética

Esta operación realiza las funciones aritméticas de Suma y Resta, entre la posición de memoria especificada y el acumulador.

02 Suma (p. ej. 021234)

El contenido de la posición especificada se suma al contenido del acumulador. El resultado de la operación se coloca en el acumulador, y el contenido de la posición de memoria permanece inalterado.

12 Resta (p. ej. 123500)

El contenido de la posición de memoria especificada se resta del contenido del acumulador. El resultado se coloca en el acu-

mulador, y el contenido de la posición de memoria permanece inalterado.

Con las dos operaciones podemos escribir un pequeño programa.

Supongamos que tenemos un 5 almacenado en la posición 0100, un 3 en la posición 0101, y un 2 en la posición 0102. Esta información puede representarse de la siguiente manera:

Dirección	Contenido
0100	00005
0101	00003
0102	00002

Queremos sumar los contenidos de las posiciones 100 y 101, restarle después el contenido de la posición 102, y almacenar el resultado (que deberá ser 6) en la posición 103.

Comenzaremos el programa en la dirección 1000, aunque podría haber sido otra cualquiera. En el programa indicado a continuación, los valores de la columna "Contenido del acumulador" corresponden al contenido del acumulador DESPUÉS de la ejecución de las instrucciones.

Dirección	Contenido (Instrucción)	Lenguaje	Contenido del Acumulador
1000	010100	Búsqueda de 100	5
1001	020101	Sumar 101	10 en octal (8 en decimal)
1002	120102	Restar 102	6
1003	110103	Almacenar 103	6
1004			

Ahora el ordenador continuará con la instrucción 1004.

Operación 3. Lógica

Esta operación trata el contenido de una posición de memoria como una representación binaria, y realiza las operaciones lógi-

cas AND o NEV (NO EQUIVALENTE) entre la posición de memoria y la representación binaria contenido en el acumulador.

03 AND (p. ej. 031021)

El contenido de la posición especificada se compara mediante "Y" con el contenido del acumulador. El resultado se coloca en el acumulador, y el contenido de la posición de memoria permanece inalterado.

Por ejemplo, la posición 1021 contiene	1001100111011101
El acumulador	0111011011000100
La instrucción 031021 dejaría	0001000011000100

en el acumulador. Esta puede parecer una instrucción caprichosa e inútil, pero sirve como máscara para extraer partes de datos o números.

13 NEV. (p. ej. 136100)

La función "No Equivalente" se aplica entre la posición de memoria especificada y el acumulador. El resultado se coloca en el acumulador, y el contenido de la posición de memoria no se altera. La función NEV resulta 1 cuando los dos dígitos comparados difieren, es decir:

1 NEV 1 resulta en 0
1 NEV 0 resulta en 1
0 NEV 1 resulta en 1
0 NEV 0 resulta en 0

Por ejemplo, la posición 6100 contiene	0011011101001100
El acumulador contiene	1011110110100101
La instrucción 136100 dejaría	1000101011101001

en el acumulador, colocando un '1' allí donde los dos bits sean diferentes. Esta instrucción, al igual que la anterior, resulta esencial como máscara. Cuando el lector llegue a dominar estas instrucciones podrá considerarse programador.

Operación 4. Desplazamiento

Las instrucciones de este tipo de operación sólo son operativas en el acumulador, y NO hacen referencia a posiciones de memoria. Su cometido consiste en desplazar la representación binaria a la derecha o a la izquierda, tantas posiciones como se haya especificado. Dado que estas instrucciones no hacen referencia a posiciones de memoria, los bits del 0 al 11 en cada una de ellas, pueden utilizarse para especificar el número de bits. Un desplazamiento de más de 16 posiciones decimales es innecesario; por tanto, la parte de "dirección" de la instrucción será de 00 a 20 en octal.

04 Desplazamiento a la izquierda (ej. 040003)

Esta instrucción desplaza a la izquierda la representación binaria contenida en el acumulador, tantas posiciones como indique la parte de dirección de la instrucción. La parte menos significativa del acumulador se llena con ceros. Supongamos que el acumulador contiene 1000111000111001. Después de ejecutar la instrucción 040002 el acumulador contendrá 0011100011100100. Es decir, 040001 multiplica por dos; 040002 multiplica por cuatro; 040003 por ocho, y así sucesivamente. No obstante, la instrucción de desplazamiento a la izquierda, a parte de multiplicar, puede realizar otras operaciones.

14 Desplazamiento a la derecha (p. ej. 140002)

Esta instrucción desplaza a la derecha la representación binaria contenida en el acumulador, tantas posiciones como se haya especificado. Como se recordará, el bit más significativo de un número binario especifica el signo (positivo o negativo). Cuando se ejecuta esta instrucción el bit de signo se mantiene y, en consecuencia, el signo del número también se mantiene. Esta operación recibe el nombre de "desplazamiento aritmético a la derecha".

Supongamos que el acumulador contiene	0000000000001101
La instrucción 140002	
nos dará el resultado	0000000000000011
Si el contenido del acumulador	
fuese un número negativo, por ejemplo el	1000000000001101
La instrucción 14002	
nos dará	1110000000000011

Una instrucción de desplazamiento a la derecha divide eficazmente por 2, 4, 8, 16, etc., pudiendo además realizar otras funciones.

Hemos de hacer la observación de que algunos ordenadores llenan con ceros el extremo izquierdo del acumulador cuando se efectúa un desplazamiento a la derecha. Este tipo de operación se conoce con el nombre de desplazamiento lógico a la derecha. Algunas máquinas permiten ambos desplazamientos; el lógico y el aritmético.

Operación 5. Salto condicional

Con las instrucciones que hemos especificado en las operaciones 1 a 4, un programa sólo puede avanzar paso a paso desde el comienzo hasta el final. El salto condicional nos permite saltar a diferentes partes del programa, dependiendo del estado del acumulador.

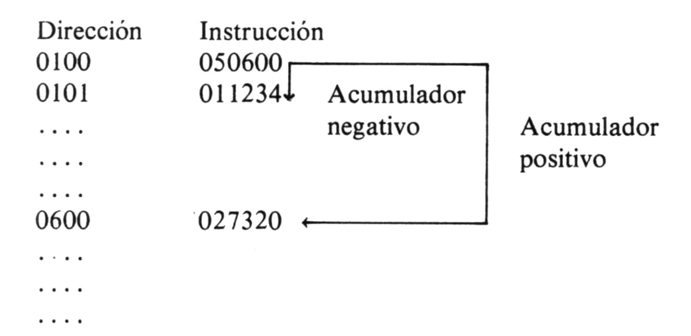
La parte de dirección de la instrucción indica la dirección de la instrucción que ha de ser ejecutada si se cumplen las condiciones especificadas. Si dichas condiciones no se cumplen, se ejecuta la instrucción que sigue al salto condicional. Esta explicación puede resultar confusa, pero las definiciones y ejemplos que se dan a continuación deberán contribuir a clarificarla.

05 Salto si es positivo (p. ej. 051735)

Si el contenido del acumulador es positivo, se ejcuta la instrucción cuya dirección está indicada en la parte de dirección. Si

el contenido del acumulador es negativo, se ejecuta la instrucción que sigue a la que está en curso.

Supongamos que tenemos la siguiente información:



Si al ejecutar la instrucción 100, el contenido del acumulador es negativo, la siguiente instrucción será la 101, seguida de la 102 y de la 103.

Si el contenido del acumulador es positivo al ejecutar la instrucción 100, se produce el salto a la posición 600, y la siguiente instrucción será la 600 seguida de la 601, 602, etc.

No debemos olvidar que el cero se considera como número positivo.

15 Salto si no es cero

Si el contenido del acumulador no es cero (es decir, si es 1 positivo o mayor, 1 negativo o menor), la siguiente instrucción a ejecutar será la especificada en la parte de dirección de la instrucción. Si el contenido del acumulador es cero, la siguiente instrucción a ejecutar será la que sigue a la que está en curso.

Supongamos que tenemos la siguiente información:

1023	154057	↓ Acumulador cero	Acumulador no cero
....			
....			
....			
....			
4057	021234 ←		
4060			

Si llegamos a la instrucción 1023 y el acumulador contiene un número que no es cero, la siguiente instrucción a ejecutar será la 4057, seguida de la 4060 (siempre en octal), la 4061 y así sucesivamente.

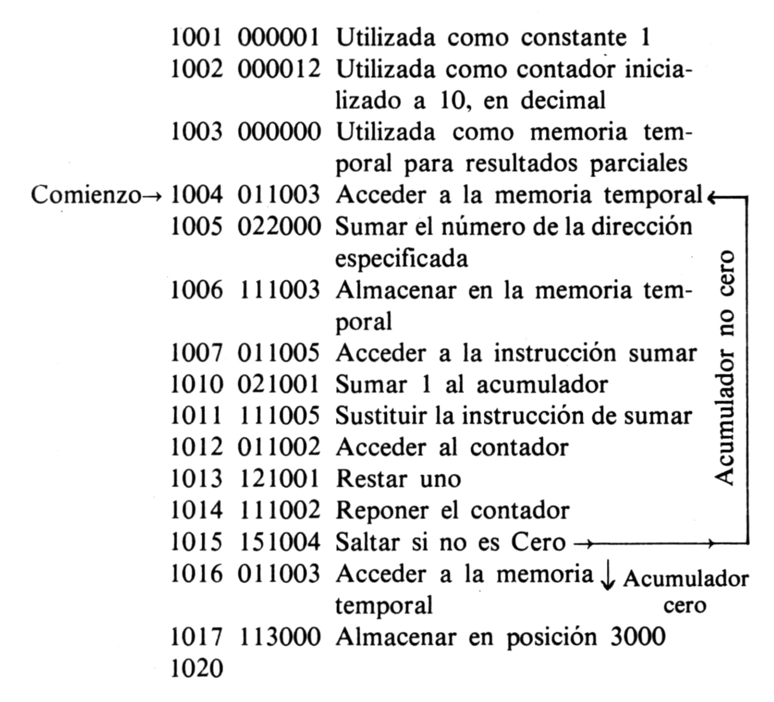
Si llegamos a la instrucción 1023 con cero en el acumulador, la siguiente instrucción a ejecutar será la 1024, seguida de la 1025, etc.

Las instrucciones de salto condicional facilitan la creación de programas flexibles y, además, permiten utilizar, una y otra vez, las partes comunes del programa. Supongamos que queremos sumar 10 números decimales que se hallan en las posiciones de memoria 2000 a 2011 (en octal), y que el resultado ha de ser almacenado en la posición de memoria 3000.

Un método rudimentario de realizar esta operación sería el siguiente:

Dirección	Instrucción	Lenguaje
1001	012000	Búsqueda 2000
1002	022001	Sumar 2001
1003	022002	Sumar 2002
1004	022003	etc.
1005	022004	
1006	022005	
1007	022006	
1010	022007	
1011	022010	
1012	022011	
1013	113000	Almacenar 3000
1014		

El programa que se muestra a continuación lleva a cabo la misma operación:



	1001	000001	Utilizada como constante 1
	1002	000012	Utilizada como contador inicializado a 10, en decimal
	1003	000000	Utilizada como memoria temporal para resultados parciales
Comienzo→	1004	011003	Acceder a la memoria temporal
	1005	022000	Sumar el número de la dirección especificada
	1006	111003	Almacenar en la memoria temporal
	1007	011005	Acceder a la instrucción sumar
	1010	021001	Sumar 1 al acumulador
	1011	111005	Sustituir la instrucción de sumar
	1012	011002	Acceder al contador
	1013	121001	Restar uno
	1014	111002	Reponer el contador
	1015	151004	Saltar si no es Cero
	1016	011003	Acceder a la memoria temporal
	1017	113000	Almacenar en posición 3000
	1020		

Este programa ilustra muchas técnicas de programación. Pero observemos particularmente su funcionamiento.

La parte del programa comprendida entre 1004 y 1015 se utiliza repetidas veces (diez en total). Esta operación se conoce con el nombre de ciclo (bucle). En la posición memoria 1002 tenemos un contaje que nos indica cuantas veces hemos recorrido el ciclo. Cada vez que lo completamos, restamos 1 del contenido de dicha posición (instrucciones 1012, 1013, 1014), y cuando el contador indique cero habremos recorrido el ciclo diez veces.

Es muy importante observar que el programa SE AUTOMODIFICA. En la primera vuelta al ciclo, la posición 1005 contiene 022000 (Suma el contenido de la posición 2000 al acumulador). Las instrucciones 1007, 1010, 1011 incrementan en uno el conte-

nido de la posición de memoria 1005. En la siguiente vuelta al ciclo, la posición 1005 contiene 022001 (Suma el contenido de la posición 2001 al acumulador). En la siguiente vuelta contiene 022002, después 022003, y así sucesivamente. Una instrucción puede ser tratada como datos por otra instrucción.

Finalmente, observamos que la posición 1003 se usa como memoria temporal de resultados parciales.

Al llegar a este punto el lector probablemente esté pensando: "Vaya manera de complicarlo", "El primer programa utilizaba menos instrucciones", "¿Por qué complicar las cosas?". La respuesta a esta pregunta reside en el número de posiciones de memoria que tiene la máquina, ya que es precisamente esto lo que limita el tamaño del programa que se puede escribir. El DIM-1 tiene 4K de memoria, pero algunos microprocesadores se comercializan con memorias de sólo 256 bits, lo que resulta bastante limitado. Para realizar diez sumas el programa simple es más corto, pero ha de tenerse en cuenta que el programa más complejo seguirá utilizando el mismo número de instrucciones para sumar diez, cien o mil números. Para ello, basta con alterar la constante en la posición 1002. Aparte del ahorro de espacio, ¿a quién le agrada escribir cien instrucciones de sumar, pudiendo utilizar el programa "más complejo" del último ejemplo?

Cuando hayamos visto la operación 6, escribiremos nuevamente el mismo problema utilizando menos instrucciones.

La mayoría de los ordenadores cuentan con saltos condicionales en su conjunto de instrucciones, pero las definiciones difieren de unos a otros. Algunos de ellos tienen "Salto si es Cero" y "Salto si es Negativo"; otros tienen los cuatro saltos condicionales posibles.

Operación 6. Aritmética unitaria

Esta función se usa para controlar ciclos como el que hemos visto, añadiendo o restando uno al contenido de la posición de memoria especificada, y dejando el resultado en la memoria Y en el acumulador.

06 Incremento (p. ej. 063725)

Esta instrucción incrementa en uno el contenido de la dirección especificada, y coloca al resultado en la posición indicada Y en el acumulador.

16 Decremento (p. ej. 166315)

Esta instrucción disminuye en uno el contenido de la dirección especificada, y coloca el resultado en la posición indicada Y en el acumulador.

Las instrucciones de incremento y decremento permiten combinar varias instrucciones en un ciclo de programa, ahorrando espacio y esfuerzo.

El programa para sumar diez números se escribe nuevamente a continuación utilizando la instrucción de incremento, y una comprobación NEV para determinar cuando está completo el ciclo.

	Dirección	Instrucción	Lenguaje
	1002	022012	Constante para comprobación NEV
	1003	000000	Memoria temporal
Comienzo→	1004	011003	Acceder a memoria temporal ←
	1005	022000	Sumar contenido de dirección especificada
	1006	111003	Almacenar en memoria temporal
	1007	061005	Incrementar instrucción de sumar
	1010	131002	NEV con nueva constante
	1011	151004	Saltar a la siguiente suma si no es cero
	1012	011003	Acceder a la memoria temporal
	1013	113000	Almacenar en 3000

Como se puede observar, ahora tenemos una máscara en la posición 1002. La instrucción de Sumar incrementada en la posición 1005 es contrastada con la máscara mediante NEV. Para cualquier número, excepto 022012, el acumulador no estará en cero y, por tanto, recorreremos otra vez el ciclo. Cuando la instrucción de sumar llegue a 022012, el resultado de la comprobación NEV será cero y entonces saldremos del ciclo para continuar con la instrucción 1012.

Este programa tiene la misma longitud que el programa rudimentario del primer ejemplo, pero, como en el caso del programa complejo anterior, tanto si sumamos diez números como si sumamos mil, su longitud es la misma.

Operación 7. Saltos incondicionales

Estas instrucciones obligan al programa del ordenador a trasladarse a una dirección distinta de la siguiente a la que está en curso.

07 Salto incondicional (p. ej. 075342)

La siguiente instrucción a ejecutar es aquella cuya dirección viene dada por la parte de dirección de la instrucción.

Por ejemplo:

Dirección	Instrucción
1234	017207
1235	023664
1236	073220
⋮	
3220	037223
3221	040001

Al llegar a la instrucción 1236, el programa efectúa un salto y, continúa con la secuencia 3220, 3221, en vez de hacerlo con las instrucciones 1237, 1240.

17 Salto y parada (p. ej. 175620)

El programa realiza un salto como en la instrucción anterior, pero antes de ejecutar la instrucción a la que ha saltado, el ordenador se detiene y no continuará hasta que se pulse la tecla de continuación. (GO) este es el modo de parar un ordenador.

Para interrumpir una secuencia de ejecución, basta con introducir (17-siguiente posición)

Por ejemplo:	5112	012345
	5113	175114
	5114	027077

El ordenador se detendrá antes de ejecutar la isntrucción 5114.

Las instrucciones de salto puede que no parezcan especialmente útiles, pero permiten escribir programas por bloques, y esto facilita las correciones o adiciones que sea necesario hacer en el programa. El uso de la instrucción de salto y parada resulta obvio.

Operación 0. Entrada y salida

Todo ordenador precisa información con la que trabajar y algún medio de dar salida a los resultados; es decir, necesita un método para la entrada de datos y otro para la salida.

Como métodos de entrada de datos comunmente empleados podríamos mencionar las lectoras de cinta perforada, contactores de relés para control de plantas industriales, interruptores de llave, etc., y como formas de salida comunes citaremos la cinta perforada, máquinas de escribir, bobinas de relé para control, luces, pantallas, etc. A veces, como medio de entrada y salida se utiliza una misma unidad, como puede ser una cinta magnética o una unidad de discos.

Por otra parte, es poco probable que un ordenador cuente únicamente con un dispositivo de entrada y otro de salida, y por

tanto hemos de poder identificar a qué dispositivo nos referimos. Para tal fin disponemos de la dirección de dispositivo.

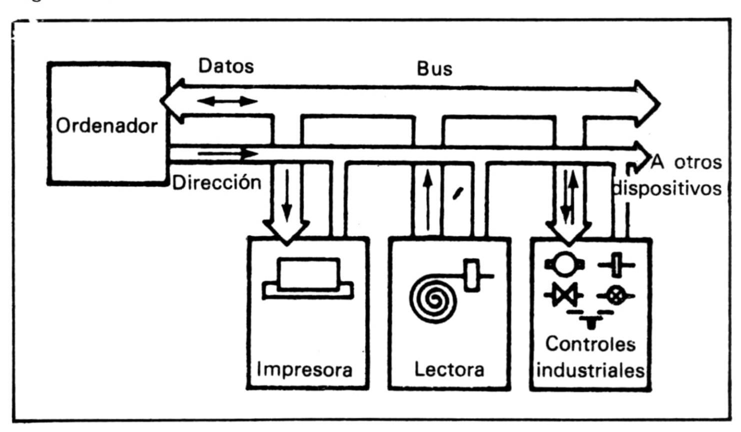
Cada dispositivo de Entrada o Salida tiene una dirección a la que responderá. El DIM-1 se comunica con los dispositivos de entrada/salida mediante una vía de transmisión (bus). Esta vía tiene 8 líneas para el tráfico de datos (llamadas conjuntamente bus de datos) y 8 líneas para la dirección del dispositivo. Las ocho líneas de datos son bidireccionales; es decir, los datos pueden entrar al ordenador o salir de él a través de una misma línea. Esto se lleva a cabo utilizando puertas de tres estados para los usuarios del bus.

Por otra parte, tenemos líneas de control con señales de sincronización como, por ejemplo, impulsos de datos y señales que identifican la entrada y salida de estos últimos.

El concepto de bus (mostrado en la figura 9) permite pasar un cable-cinta de dispositivo en dispositivo, y constituye una de las peculiaridades de los ordenadores modernos.

Por consiguiente, para realizar una transferencia de I/O hemos de identificar la siguiente información:

Fig. 9 Vía de transmisión del ordenador.



a. La dirección del dispositivo (8 bits)
b. Los datos (8 bits)

Como el lector habrá adivinado, la dirección del dispositivo estará contenida en la parte de dirección de la instrucción, pero no deberá confundir la dirección del dispositivo de I/O con la dirección de memoria.

00 Entrada (p. ej. 000177)

La dirección del dispositivo seleccionado se encuentra en la parte de dirección de la instrucción, y ha de estar comprendida entre 0000 y 0377.

Los datos se cargan en los 8 bits de orden inferior del acumulador.

10 Salida (p. ej. 100201)

Como en el caso anterior, la parte de dirección especifica el dispositivo seleccionado. Los datos reciben salida desde los 8 bits de orden superior del acumulador, desplazándose éste 8 bits a la izquierda. Esto simplifica la transferencia de los datos en 16 bits y la de los contenidos en octetos (bytes) sucesivos.

La selección de los 8 bits de la derecha para la entrada y los 8 de la izquierda para la salida es deliberada. Normalmente, la representación de datos se hace comenzando por la parte más significativa (p. ej. al escribir 327). Este procedimiento nos permite tomar dos bloques de datos y terminar con ellos ordenados correctamente en el acumulador. Del mismo modo, dos bloques de datos contenidos en un acumulador recibirán salida en el orden correcto.

Algunos ordenadores son más flexibles que el DIM-1 en el sentido de que cuentan con dos buses de 16 bits; uno para los datos, y otro para la dirección. En tal caso, para efectuar una salida se necesita una dirección de 16 bits, y una señal de permiso

funcional del dispositivo y la pseudoinstrucción ("comando") de I/O. Por otra parte, hay máquinas dotadas de instrucciones de I/O muy complejas que permiten la transferencia directa de datos entre la memoria y los dispositivos de I/O.

En general, la utilidad de un ordenador depende de la flexibilidad de sus instrucciones de I/O. El conjunto de instrucciones de I/O del DIM-1 es muy limitado, pero, como es lógico, muy simple.

El mayor problema que se encuentra en la entrada y salida de datos lo constituye la enorme diferencia de velocidad existente entre el ordenador y los dispositivos mecánicos a él conectados. Generalmente, los dispositivos mecánicos emiten una señal cuando han terminado la transferencia de los datos (p. ej., cuando la impresora ha terminado de imprimir o la lectora ha presentado el carácter al ordenador), que es utilizada para sincronizar los dispositivos de I/O y el ordenador.

La manera más simple de armonizar estas diferencias de velocidad consiste en aceptarlas. El ordenador podría sacar resultados a una impresora y después, metafóricamente hablando, "sentarse a descansar" hasta que vuelva a recibir la señal de la impresora indicándole que ha terminado de imprimir. El ordenador continuaría, entonces, con la siguiente instrucción del programa.

Obviamente, esto significaría desaprovechar velocidad de cálculo, ya que hay varios métodos para resolver el problema. Uno de ellos consiste en utilizar un conjunto de TTL (o CMOS), asociado al dispositivo de I/O (generalmente alguna forma de control y cerrojo de 8 bits). Esta lógica funciona a velocidad de ordenador, de modo que las transferencias entre este último y la lógica no ocasionan la detención del programa. Entonces, la lógica excita al dispositivo de I/O.

Por otra parte, esta función lógica genera una palabra de 8 bits (llamada "palabra de estado") que indica la situación del dispositivo y que puede ser leída por el ordenador a gran velocidad. En la jerga del sector se dice que un dispositivo está "ocupado"

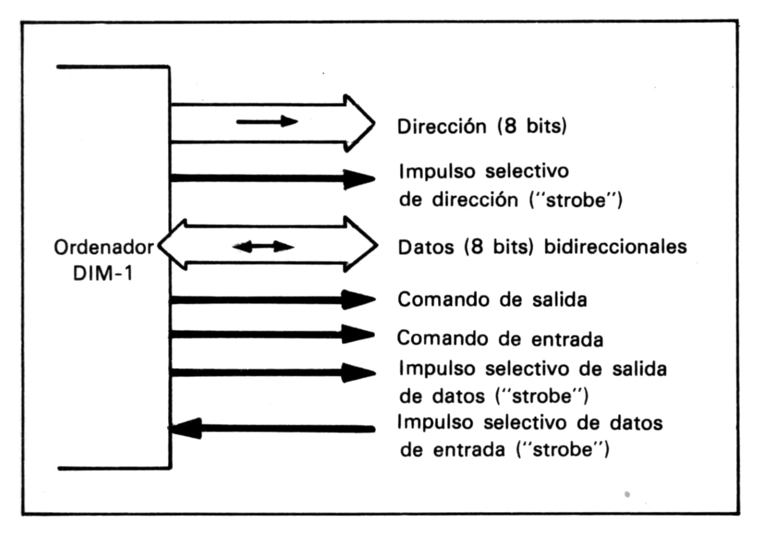


Fig. 10 Vía de transmisión de I/O del DIM-1.

cuando no puede aceptar un "comando", y que está "listo" cuando puede aceptarlo.

Para ilustrar esta explicación utilizaremos una impresora con la dirección de dispositivo 60, y una palabra de estado con la dirección 61. La figura 10 muestra la vía conjunta de entrada y salida. Utilizando puertas lógicas de tres estados podemos usar una sola vía (bus) para efectuar la entrada y la salida de datos. Tenemos varias líneas de control; impulso selectivo de dirección ("strobe"), comando de salida, impulso de datos de salida, y comando de entrada que son generados por el ordenador, y un impulso de entrada de datos generado por el dispositivo.

Ahora ejecutemos la instrucción 100060 (salida al dispositivo 60). Lo primero que hace el ordenador es leer la palabra de estado. La lógica está preparada de manera que dicha palabra sea cero si el dispositivo está listo, y '1' (todos "unos") si está ocupado.

El programa comienza de la siguiente manera:

```
1000 000061    (lectura del estado)
1001 151004    (salto si no es cero; es decir, dispositivo ocupado)
1002 01NNNN Carga de datos desde NNNN
1003 100060    Salida a la impresora
1004 continuar
```

La sincronización de la transferencia es de gran interés, y se resume en la figura 11. La secuencia operativa detallada de la instrucción de salida en curso es el siguiente:

a. La dirección se introduce en la línea de dirección.
b. Después de un breve intervalo, se envía un impulso selectivo de dirección ("strobe") que "activa" la lógica del dispositivo seleccionado. Normalmente, el dispositivo envía al ordenador otra señal como respueta, pero ésta se omite en el DIM-1. Después del impulso de dirección, ésta es eliminada.
c. Los datos se colocan en el bus de datos, junto al "comando" de salida.
d. Después de un pequeño intervalo se envía el impulso selectivo de salida de datos ("strobe"). Los datos quedan retenidos en la lógica del dispositivo y se extraen del bus.
e. El ordenador pasa a ejecutar la siguiente instrucción, mientras la lógica del dispositivo excita los solenoides, libera embrague y realiza otras operaciones encaminadas a poner en funcionamiento al dispositivo.

La entrada, paso a paso, de la palabra de estado sería:

a. Se coloca la dirección en las líneas de dirección, y se envía un impulso selectivo.
b. Se envía un "comando de" entrada.
c. El dispositivo coloca los datos en el bus de datos y, después de un breve intervalo, envía un impulso selectivo de "entrada de datos". De esta manera los datos se introducen en el ordenador.

Cuando se usa una lectora de cinta, son necesarias varias entradas y salidas:

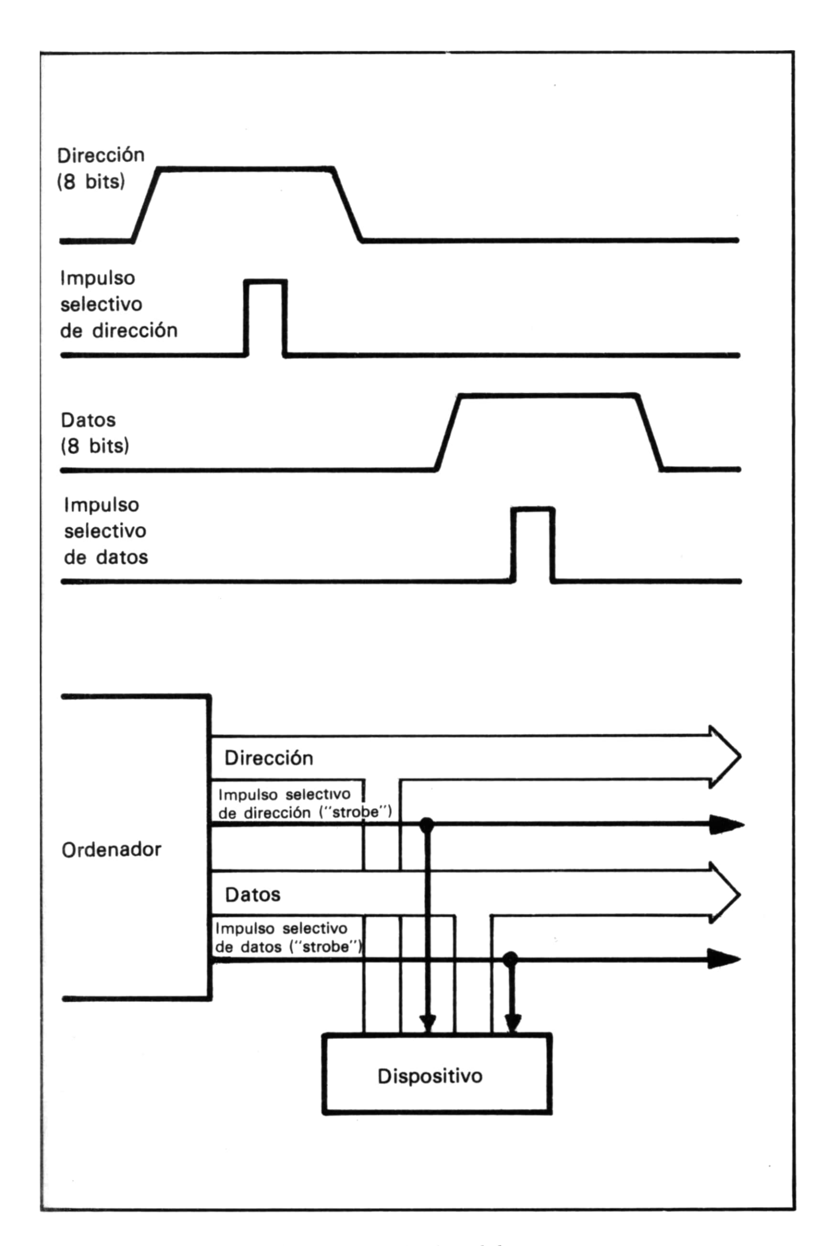


Fig. 11 Sincronización de un comando de salida.

a. Se lee una palabra de estado para comprobar si la lectora está lista.
b. Se hace una salida para activar la lógica (que levanta enganches, arranca motores y lee un carácter colocándolo en el almacenamiento intermedio lógico).
c. Se hacen una o más entradas de la palabra de estado para determinar cuando ha sido leído el carácter desde la cinta al almacenamiento intermedio lógico.
d. Se hace una última lectura de dos datos cuando la palabra de estado indica que están preparados para ser introducidos en el ordenador.

Sobre el tema de entrada y salida existen muchas variantes, y volveremos a tratarlo más adelante cuando hablemos sobre las unidades UART y pastillas ("chips") de PIO (I/O en paralelo).

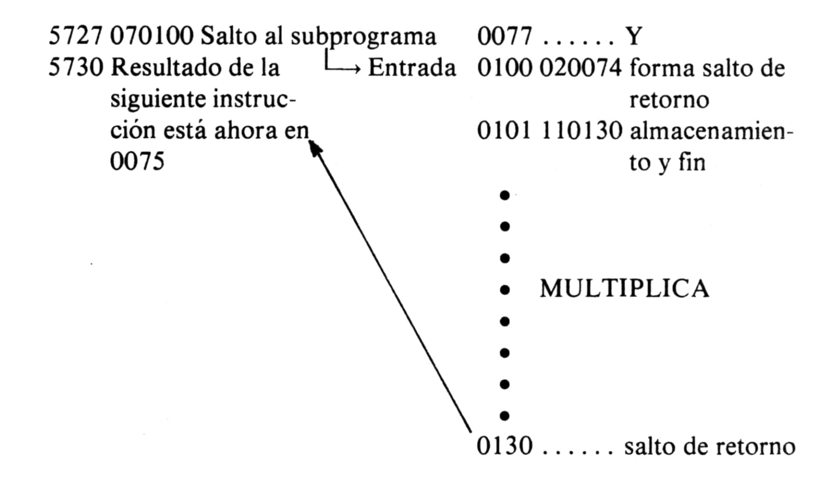
5. SUBRUTINAS (SUBPROGRAMAS)

Apuntábamos anteriormente que las funciones de multiplicación y división no se podían escribir en el programa. Pero sería muy útil incluirlas por una sola vez y dejar que, por ejemplo, la rutina de "multiplicación" sea utilizada por cualquier parte del programa que necesite realizar una multiplicación. A esta técnica se le llama escritura de subrutinas o de subprogramas.

Supongamos que escribimos un programa para multiplicar, y que comenzamos en la posición 0100. Los dos números objeto de multiplicación se encuentran en las posiciones 0077 y 0076, y el resultado se almacena en 0075. (Esta es una manera de simplificar las cosas, ya que la multiplicación de dos números de 16 bits da como resultado 32 bits. Pero para mayor sencillez, digamos que los dos números que vamos a multiplicar no pueden exceder de 8 bits.) La posición 0074 contiene la constante especial 06002.

PROGRAMA PRINCIPAL
Coloca X en 0076
Coloca Y en 0077
5726 015726 Busca esta posición y coloca su contenido en A

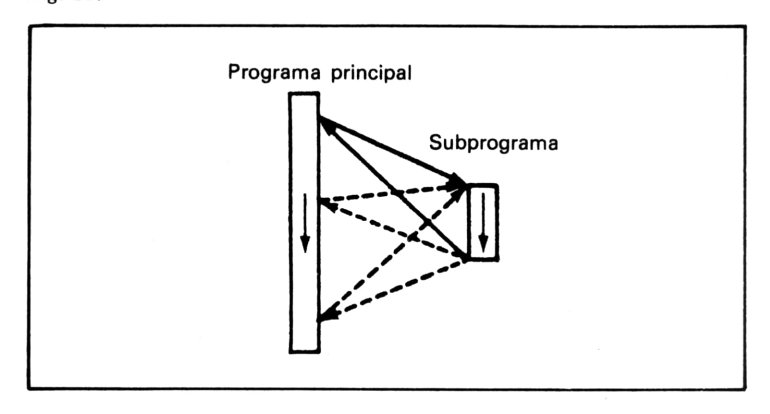
SUBPROGRAMA
0074 060002 constante de retorno
0075 . . . . . . Resultado
0076 X



El programa principal coloca X en 0076 e Y en 0077, y después la instrucción 5726 carga en A el contenido de 015727. La siguiente instrucción salta al subprograma (070100).

La primera instrucción que aparece en el subprograma añade 060002 al acumulador formando 075730. Este resultado es almacenado en la última instrucción del subprograma (posición 0130). 075730 es una instrucción de salto de retorno a la siguien-

Fig. 12.



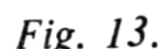
te instrucción en el programa principal. Así pues, el subprograma realiza la multiplicación y retorna al programa principal.

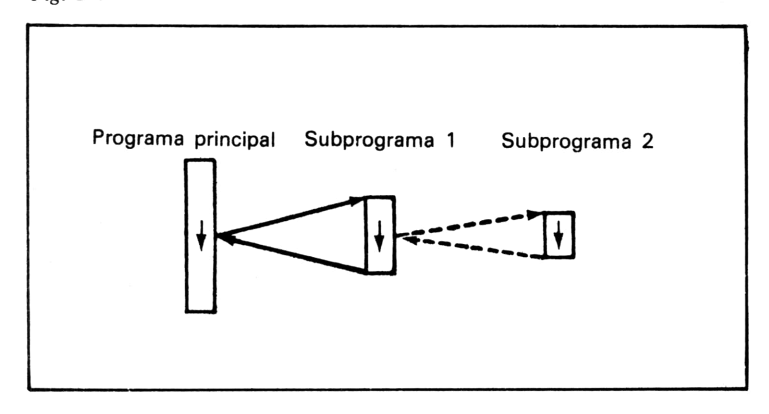
A la estructura del salto de retorno establecida antes de bifurcar a un subprograma se le suele llamar "Implantación del enlace".

Al lector le alegrará saber que en los microprocesadores el funcionamiento de subprogramas es mucho más simple. Normalmente hay una instrucción "Llamar subprograma", que guarda la dirección de la instrucción en curso y salta a la dirección del subprograma especificado. Al final del subprograma se coloca una instrucción de retorno ("Return"), que recupera la dirección de instrucción y retorna al programa principal. Este procedimiento es, por supuesto, mucho más simple que nuestra demoledora técnica.

Es deseable que un subprograma pueda llamar a otro. Por ejemplo, el programa principal podría llamar un subprograma para convertir un número binario a código BCD. Pero el subprograma de binario BCD necesitará llamar a otro subprograma de división para realizar la conversión.

Cuando un subprograma va incluido en otro se dice que está "encajado".


Fig. 13.



6. ENSAMBLADORES, AUTOCODIGOS Y LENGUAJES DE ALTO NIVEL

Con las 16 instrucciones del DIM-1 se pueden realizar casi todas las funciones de programación pero, como veremos, la escritura de un programa sería una pesada tarea. En realidad, las fases de creación de un programa se apartan un poco del propósito de este libro, pero básicamente son las siguientes:

a. Identificar el problema y "comprenderlo".
b. Fraccionarlo en pequeños pasos que puedan ser identificados.
c. Trazar un organigrama de cada paso.
d. Escribir las instrucciones de ordenador correspondientes a los organigramas.

Los programas que hemos visto hasta ahora estaban escritos en lenguaje que el ordenador puede comprender. A esto se le llama programación en código máquina, y ocupa el nivel más bajo en la jerarquía de lenguajes de programación.

La programación en código máquina es muy aburrida, y plantea algunos problemas que se presentan al intentar depurar los programas. Por ejemplo, supongamos que hemos escrito un programa que ocupa unas 4K de memoria y que al probarlo detectamos la omisión de una instrucción en medio del programa. Para nuestra desesperación, tendremos que desplazar 2K de instrucciones una posición hacia abajo. Peor aún, tendremos que modificar cantidad de instrucciones debido a que las partes de

dirección habrán cambiado. Evidentemente, por este camino pronto llegaremos al ataque de nervios.

Otro problema se refiere a la comprensión. En el momento de escribir un programa resulta fácil de comprender, pero ¿qué ocurriría después de 6 meses? Las instrucciones del DIM-1 son lógicas, pero cuando tuviésemos que enfrentarnos de nuevo a un programa de 4K, escrito únicamente en código máquina, ¿quién podría enterarse de algo?

Si disponemos de un ordenador equipado con lectora de cinta, teletipo y perforadora, la programación puede realizarse con mayor facilidad. En tal caso, podemos utilizar un programa de ordenador llamado Ensamblador (suministrado por el fabricante) para simplificar la escritura y depuración del programa en código máquina. Veamos cómo nos ayuda un Ensamblador.

En primer lugar, para cada una de las instrucciones que se muestran a continuación utilizamos un código mnemotécnico de tres letras:

Acceder al acum.	FCH
Almacenar del acum.	STR
Sumar al acum.	ADD
Restar del acum.	SUB
Y con acum.	AND
No equivalente con acum.	NEV
Desplazar a izquierda	SHU
Desplazar a derecha	SHD
Saltar si es positivo	JPE
Saltar si no es cero	JNZ
Incrementar	INC
Decrementar	DEC
Saltar	JMP
Saltar y parar	STP
Introducir	INN
Hacer salir	OUT

A continuación definimos etiquetas. Una etiqueta es un código mnemotécnico (mnemónico, en la jerga) que se puede asignar

a una posición de memoria para su identificación. En esta ocasión, el código mnemotécnico consiste en una palabra de cuatro letras (p. ej. FRED, TEMP, FAIL) que podemos utilizar para identificar datos o instrucciones. El programador puede elegir estas palabras libremente.

El Ensamblador del DIM-1 permite escribir programas en código máquina utilizando mnemotécnicos y etiquetas. Además, cualquier instrucción o dato escrito en octal es aceptado por el ordenador sin conversión. De este modo, las máscaras y datos similares pueden escribirse directamente.

Por ejemplo, la suma de diez números efectuada con el programa que utilizábamos al definir la instrucción de Decrementar, podría escribirse de la siguiente manera:

```
         Etiqueta   Instrucción
         MASK       022012
         TEMP       000000
Inicio → STRT       FCH        TEMP
         ADDN       ADD        DATA ←──────┐
                    STR        TEMP        │
                    INC        ADDN        │Salto
                    NEV        MASK        │NZ
                    JNZ        ADDN  ──────┘
                    FCH        TEMP
                    STR        SOLN
```

Queda claro que este programa resulta más fácil de escribir y comprender. En primer lugar, no hemos definido la posición de memoria a la que van las instrucciones, porque el ensamblador lo hace por nosotros. En segundo lugar, no tenemos que definir etiquetas para cada una de las instrucciones. Pero, generalmente, sí hemos de definir una etiqueta por cada posición de memoria cuyos datos sean objeto de acceso de una instrucción.

En este programa hay dos etiquetas sin definir; DATA (que es la posición del primer número a sumar; 2000) y SOLN (que es la posición en la que ha de colocarse el resultado; 3000). Estas etiquetas pueden definirse en cualquier parte del programa, o al

comienzo del programa en lenguaje ensamblador, donde, el programador suele codificar las "operaciones auxiliares" y define los nombres de etiquetas.

El programa escrito en lenguaje ensamblador se llama "Programa Fuente". Para poder usar el lenguaje de ensamblaje es preciso cargar en el ordenador un programa llamado Ensamblador que, generalmente, es suministrado por el fabricante (a un precio determinado) en un disco, cassette o cinta de papel. Después se carga el programa fuente bajo el control del Ensamblador.

Seguidamente, el Ensamblador convierte el programa fuente en código máquina, y genera el correspondiente programa en dicho código. Esta operación es bastante directa, ya que existe una correspondencia biunívoca entre mnemotécnicos y códigos de instrucción, y entre etiquetas y direcciones.

El programa en código máquina obtenido es perforado en cinta (o grabado en disco o cassette), y por medio de una impresora se obtiene un documento con un listado del programa. Este documento lo necesitaremos para seguir el programa más tarde. El programa en código máquina se llama "Programa Objeto" y se vuelve a cargar en el ordenador para que realice la tarea que le ha sido asignada.

Algunos ordenadores tienen memorias con suficiente capacidad para alojar al mismo tiempo el ensamblador y el programa objeto, y no es necesario obtener y volver a cargar una cinta objeto. De este tipo de sistemas se dice que tienen un ensamblador residente.

Es importante aclarar que el ensamblaje puede realizarse en un ordenador distinto (de otro fabricante) al utilizado para ejecutar el programa objeto. Por ejemplo, es posible obtener un ensamblador Z80 que reside en un DEC PDP11-34. Los programas fuente para el Z80 se cargan en el 11-34, y éste produce un programa objeto que después se carga en el microprocesador Z80.

La mayoría de los ensambladores tienen más funciones que las que hemos descrito hasta ahora. Como función típica podría-

mos citar, la que nos permite escribir nombres mnemotécnicos como GOSUB MULT, que implanta automáticamente el enlace y salta al subprograma MULT. Al final del subprograma, RET nos llevaría de vuelta al punto de partida.

Es corriente añadir un "decalaje" a las etiquetas, así por ejemplo:

```
IMP        STRT        + 7
```

significa saltar a la posición situada siete posiciones a continuación de la indicada mediante la etiqueta STRT.

Algunos ensambladores utilizan el asterisco * para indicar "esta posición", de modo que:

```
JNZ        * – 5
```

significa "saltar cinco posiciones hacia atrás, si el acumulador contiene un número distinto de cero".

Del mismo modo:

```
ADD        * + 4
```

significar "sumar al acumulador el contenido de la posición cuatro, contando desde la posición actual".

El uso del lenguaje ensamblador reporta muchas ventajas; los nombres mnemotécnicos son fáciles de recordar y, por consiguiente, los programas también lo son de escribir. No obstante, la principal ventaja reside en que un programa fuente es fácil de modificar. Si deseamos insertar una instrucción adicional sólo tenemos que cargar el programa fuente hasta el punto de la inserción, cargar la instrucción adicional y, después, el resto del programa fuente. Con esta operación, el ensamblador produce un programa objeto totalmente nuevo, incluyendo documentación, en cuestión de segundos.

Desafortunadamente, pocos son los principiantes que pueden disponer de la memoria, lectora y perforadora necesarias para

operar un ensamblador (y mucho menos el propio Ensamblador), pero, de todos modos, deberán escribir programas en lenguaje ensamblador, convirtiéndolos luego manualmente a código máquina. Mediante este método se obtienen programas bien documentados que pueden ser comprendidos aunque haya transcurrido tiempo. Un programa documentado tendrá un aspecto similar al siguiente:

Posición	Código Objeto	Etiqueta	Código Fuente	Comentarios
1002	022012	MASK	022012	Ultima instruc. de sumar
1003	000000	TEMP	000000	Suma acumulativa
1004	011003	STRT	FCH TEMP	Comienzo del programa
1005	122000	ADDN	ADD DATA	Añade siguientes datos
1006	111003		STR TEMP	Almacena en suma acumulativa
1007	061005		INC ADDN	Incrementa dirección de suma
1010	131002		NEV MASK	Comprobación para último número
1011	151004		JNZ ADDN	Salto si no es cero
1012	011003		FCH TEMP	Acceso al resultado
1013	113000		STR SOLN	Almacena solución

Fijémonos en los comentarios. Estos comentarios explicativos son realmente muy útiles cuando transcurridos varios meses desde que hemos escrito y olvidado el programa deseamos utilizarlo de nuevo.

Los Ensambladores que hay en el mercado tienen distintos grados de inteligencia. Los más complejos llevan a menudo rutinas incorporadas. Por ejemplo, si quisiéramos sumar LUIS y PACO, y colocar el resultado en JUAN (estos nombres son etiquetas), escribiríamos lo siguiente:

```
FCH    LUIS
ADD    PACO
STR    JUAN
```

Un ensamblador inteligente nos permitiría escribir:

ADD LUIS, PACO, JUAN

Si quisiéramos sumar LUIS y PACO, y colocar el resultado en LUIS, escribiríamos:

ADD LUIS, PACO, LUIS.

En cada caso, el ensamblador incluiría por nosotros en el programa objeto las instrucciones FCH, ADD y STR necesarias.

A los ensambladores complejos a veces se les llama autocódigos, aunque los límites entre un ensamblador, un autocódigo y un lenguaje de alto nivel son muy vagos.

Los lenguajes de alto nivel se sitúan a la cabeza en la jerarquía de programas, y permiten escribir programas casi como lo haríamos en un idioma cualquiera (inglés, español, etc.). Por ejemplo, una sentencia típica en lenguaje BASIC sería:

```
100 LET WATTS = CURR↑ 2 × RES
110 IF WATTS > 2 THEN 200
```

Esta sentencia, similar al idioma inglés, evalúa la conocida fórmula eléctrica $W = I^2 R$, y, después, comprueba si el número de vatios es mayor que 2.

El lenguaje de alto nivel es, por supuesto, un programa en código máquina que reside permanentemente en un ordenador. El programa fuente se carga en el ordenador, y es convertido a lenguaje máquina. Pero, a diferencia de lo que ocurre con los ensambladores y autocódigos, no se genera cinta objeto, y, generalmente, el programador nunca ve el programa en código máquina. Los lenguajes de alto nivel suelen utilizarse para cálculos científicos y empresariales, ya que en estos campos la escritura de programas en código máquina resulta excesivamente trabajosa, incluso con la ayuda del lenguaje ensamblador.

Como lenguajes de alto nivel típicos citaremos el Fortran, Algol, Coral y el famoso Basic. Este último es uno de los lenguajes de programación más fáciles de aprender, existiendo versiones que caben en 2K de memoria. Esto permite escribir programas en BASIC en microprocesadores bastante pequeños.

Aparte del aspecto familiar de sus instrucciones, los lenguajes de alto nivel son notables por su amplia gama de funciones; todos tienen las cuatro funciones aritméticas con múltiples opciones, la mayoría cuenta con exponenciación y raíz cuadrada, y muchos de ellos disponen de funciones geométricas y estadísticas. La sencillez de programación es, no obstante, la característica común de todos los lenguajes de alto nivel.

Hemos de dejar claro que estos lenguajes no están relacionados con una máquina determinada. Un programa escrito en Basic para ejecutarlo en un Z80 con lenguaje Basic, también podrá ser ejecutado en un POP11-34 o en un ICL 1902, dotados de Basic. El programa en código máquina cargado originalmente en los ordenadores para que éstos puedan aceptar programas fuente en Basic será, por supuesto, totalmente diferente. Pero los programas fuente en Basic deberan ser idénticos.

7. EL DIM-1 AMPLIADO

El DIM-1 es un ordenador muy simple, y a poco que lo utilicemos nos daremos cuenta de sus limitaciones. Probablemente, la primera modificación que le haríamos sería la de aumentar su número de registros. Un sólo registro en el DIM-1 significa que hemos de estar continuamente almacenando datos en la memoria y sacándolos de ella, incluso con los programas más insignificantes.

Así pues, sin más rodeos, incrementaremos el número de registro a 8, les asignaremos las denominaciones A, B, C, D, E, I, IR y SP.

Los cinco primeros (A – E) son registros para fines generales, como el acumulador que hemos venido utilizando, mientras que I, IR y SP son registros especiales sobre los que volveremos pronto.

Al utilizar ocho registros necesitamos modificar el conjunto de instrucciones (repertorio de instrucciones en la jerga). Ahora, cada instrucción tiene que especificar la operación, el registro y la dirección de memoria. Pero resolvamos nuestros problemas de uno en uno. Para especificar 8 registros necesitamos tres bits. Utilicemos entonces los bits 11, 10 y 9, de la siguiente manera:

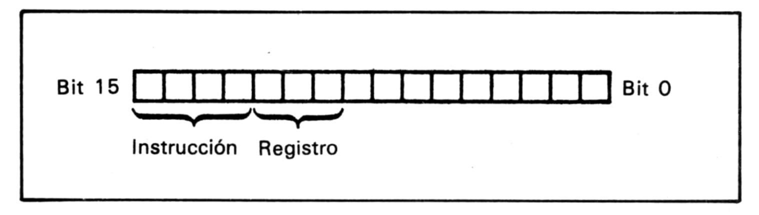


Fig. 14.

Ahora podemos decir: "Acceder al registro N" o "Sumar al registro C", y así sucesivamente para todas las instrucciones.

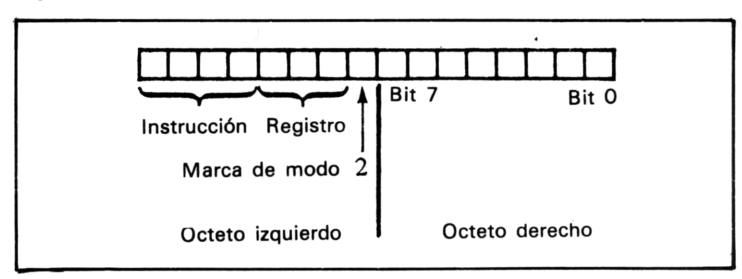
Sin embargo, esto nos crea el problema de que sólo nos quedan los bits 0-8 para especificar la dirección, y con ellos no podemos direccionar 4K de memoria. Ante tal situación, tenemos tres opciones:

a. Limitarnos a utilizar las 512 posiciones que podemos direccionar.
b. Ampliar la capacidad de nuestra máquina a 19 bits con el fin de poder utilizar 12 para la dirección.
c. Hallar cualquier otro método de direccionamiento.

Veamos lo que ocurre seleccionando la opción C. Seamos optimistas y digamos que queremos direccionar no 4K, sino 64K de memoria. ¡Ya que tenemos un problema, dejemos que sea grande!

Dos soluciones simples son las llamadas modo de página y modo relativo. Para comprenderlas, fraccionemos la dirección de 16 bits en dos octetos (bytes) de 8 bits:

Fig. 15.



El octeto izquierdo (de mayor peso) contiene 7 bits que describen la instrucción, y un nuevo bit señalizador. El octeto derecho (de menor peso) puede representar un número del 0 al 255 (ó – 128 + 127, si el séptimo bit se usa como bit de signo).

El señalizador 2 se emplea para definir el modo de direccionamiento; si es un '1', estamos usando el modo relativo, y si es un '0', el modo de página.

Explicaremos primero el modo relativo, por ser el más simple. Los 8 bits de dirección son un desplazamiento que ha de ser sumado (o restado) al valor actual del contador del programa. Esta función resulta útil dado que las instrucciones suelen hacer referencia a posiciones bastante cercanas.

Supongamos que nos encontramos en la instrucción 163220 (recuerde que ahora estamos direccionando 64K y que, por consiguiente, las direcciones estarán comprendidas entre 000000 y 177777). Ejecutamos la instrucción "Sumar a C relativo +101", que añadirá el contenido de la posición de memoria 163321 al registro C (163220 + 101 = 163321).

Este modo es particularmente útil para saltos condicionales e incondicionales, porque nos permite decir cosas como: "Saltar avanzando 23 instrucciones" o "Saltar retrocediendo 5 instrucciones", sin necesidad de saber la dirección de las instrucciones.

El modo relativo permite la escritura de programas que, cargados en cualquier parte de la memoria, se ejecutarán de manera automática. Los subprogramas comunes (como multiplicar), que el programador suele utilizar, se escriben a menudo totalmente en modo relativo, al objeto de poder "conectarlas" cuando sean necesarias.

Veamos ahora el modo de página. Dado que podemos utilizar los bits del 0 al 7 para direccionar 256 posiciones (en decimal), dividimos las 64K en páginas de 256 posiciones cada una. De esta manera, la página cero contiene las posiciones 0 a 255, la página uno de 256 a 511, y así sucesivamente.

En octal, cada página contiene 400 posiciones:

Página 0	0 – 377
Página 1	400 – 777
Página 2	1000 – 1377
Página 3	1400 – 2000

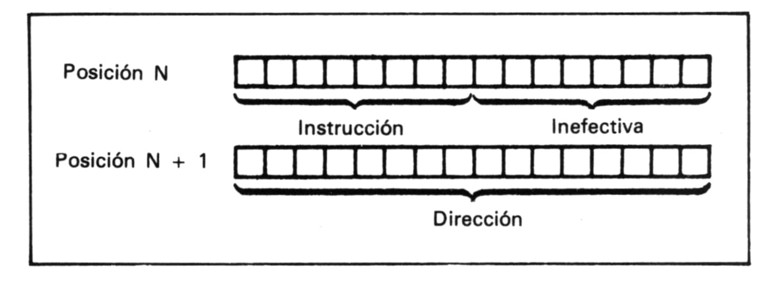
etc.

En el modo de página, el octeto derecho de la instrucción indica la posición dentro de la página en ejecución. Por ejemplo, supongamos que estamos en la posición 001532 y que la instrucción es: "Almacenar D Modo de página 171." Como la posición 001532 se encuentra en la página tres, se suma 171 a 001400 y se obtiene la posición 001571. Es decir, esta instrucción almacena el contenido del registro D en la posición de memoria 001571.

Hemos visto que tanto el modo relativo como el de página nos permiten acceder —no sin dificultad— a todas las posiciones de una memoria de 64K. También habremos notado, que para acceder a las páginas adyacentes a la que está en curso de ejecución es preciso utilizar el modo relativo. Por consiguiente, dado que el direccionamiento es todavía un poco difícil, trataremos de encontrar otros métodos.

Ante todo, recordemos que son necesarios 16 bits para direccionar 64K. Una técnica simple, aunque un tanto rudimentaria, consiste en emplear DOS palabras para direccionar la memoria. La primera palabra contiene la función (y sobra un octeto), y la segunda la dirección:

Fig. 16.



Este método se conoce con el nombre de direccionamiento directo, y, en realidad, es una extensión del esquema inicial del DIM-1. Hemos conseguido una instrucción que en vez de ocupar una posición ocupa dos. La parte de función de la instrucción ha de tener algún señalizador para indicar al control la longitud de la instrucción.

El siguiente método se conoce como indirecto de registro. Como los registros tienen una longitud de 16 bits, pueden contener una dirección de la memoria de 64K. De esta manera podemos decir algo parecido a:

> "Sumar al registro A el contenido de la posición cuya dirección se encuentra en el registro C."

Aunque sea necesario leer la sentencia varias veces para comprenderla, no debemos desmoralizarnos. Para representar este modo operativo es costumbre utilizar paréntesis delimitándolo. Por ejemplo, esta sentencia se representaría en lenguaje ensamblador de la siguiente manera:

Sumar A, (C), A

Esto significa sumar el registro A al contenido de la posición cuya dirección se encuentra en el registro C, y colocar el resultado en el registro A.

El direccionamiento indirecto de registro es muy útil cuando es necesario acceder a varias posiciones seguidas. Por ejemplo, nuestro programa anterior para sumar diez números podría escribirse de la siguiente manera:

	Posición	Contenido	
	1000	002000	Dirección del primer número
	1001	000010	Contaje
Introducir →	1002	Acceder C	posición 1000
	1003	Sumar A, (C), A. Suma A y el contenido de la dirección en C	
	1004	Decrementar contaje, resultado B	
	1005	Incrementar 1000 resultado C	
	1006	Saltar B NCero relativo – 3	
	1007	Almacenar A en 3000 (resultado)	

Obsérvese que este programa es un poco simbólico. Como todavía no hemos tratado la codificación de los distintos modos de direccionamiento, es posible que algunas instrucciones sean el doble de largas. No obstante, este simbolismo nos servirá para demostrar el uso de dichos modos.

El siguiente tipo de direccionamiento no accede a la memoria. Con varios registros a menudo necesitaremos desplazar datos de registro en registro, o realizar funciones tales como: "registro A más registro D". A este tipo de direccionamiento se le llama simplemente direccionamiento de registro. Los nombres mnemotécnicos abarcan todas las instrucciones existentes. Los siguientes son ejemplos típicos:

Add A, B, A
Fetch A, C
And D, E, D

El formato es el usual; fuente, fuente, resultado. El direccionamiento de registro y el indirecto de registro no deben confundirse. La instrucción.

Add A, C, A
y Add A, (C), A

son totalmente diferentes.

Tampoco el siguiente modo de direccionamiento accede a la memoria. Con frecuencia queremos cargar un registro con datos fijos; por ejemplo, para extraer parte de los datos en él contenidos, o para sumarle o restarle una constante. Un modo de direccionamiento llamado "Inmediato" ejecuta la instrucción utilizando los datos especificados. Así, la instrucción:

"Acceder A inmediato 256" coloca 256 en A
y "Sumar C inmediato 30" suma 30 al contenido de C

Es obvio que el modo de direccionamiento inmediato sólo es válido para determinadas instrucciones. Por ejemplo, "Almacenar A inmediato 118" no tiene significado.

En la lista de registros que añadíamos al DIM-1, había uno con etiqueta IR. Este es el conocido como registro índice y se usa en el modo de direccionamiento que vamos a tratar a continuación. El registro índice tiene 16 bits de longitud y puede, por consiguiente, contener una dirección completa de 16 bits. En el modo de direccionamiento llamado "indexado", la instrucción contiene un desplazamiento (positivo o negativo) que se añade al número en el registro índice para obtener la dirección. Este método es, por tanto, una combinación de direccionamiento relativo y direccionamiento indirecto de registro.

Supongamos que el registro índice contiene 003000, y que tenemos la instrucción siguiente:

Str A Indice + 150

El ordenador almacenará el contenido de A en la posición de memoria 3150.

Dado que el registro índice es en sí un registro de operación, puede ser utilizado como cualquier otro, y la "dirección de base" puede ser cargada, cambiada y modificada.

Tenemos, pues, varios modos de direccionamiento, y el lector decidido a asimilarlos probablemente estará un tanto confuso. Los modos que hemos visto son los siguientes:

Directo	– La instrucción contiene los 16 bits de la dirección a la que se hace referencia.
Relativo	– La instrucción contiene un desplazamiento que, sumado a la dirección en curso del programa, permite obtener la dirección a la que se hace referencia.
Indirecto de registro	– La instrucción indica un registro que contiene los 16 bits de la dirección a la que se hace referencia.
De registro	– La instrucción se ejecuta únicamente entre registros.
De página	– La instrucción contiene un número que indica la dirección de referencia dentro de la página en ejecución.

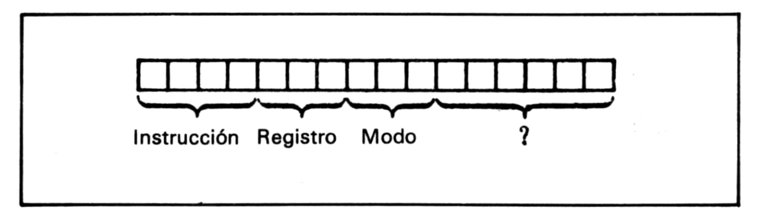
Indexado — La instrucción contiene un número que se añade al contenido del registro índice para obtener la dirección de referencia.

Inmediato — La instrucción opera de inmediato con los datos contenidos en la instrucción.

Estos siete modos de direccionamiento permiten al programados escribir programas compactos y claros, eligiendo para cada instrucción el modo que mejor se ajuste a la aplicación.

El lector probablemente se habrá dado cuenta de que se ha evitado tratar la codificación en binario de estos modos de direccionamiento. Si intentamos realizar una codificación lógica pronto tendremos problemas, debido a que los 16 bits de una palabra son insuficientes. Por ejemplo, un intento de esta operación podría ser el siguiente:

Fig. 17.



Los 6 bits restantes serían justos para los modos de direccionamiento de registro e indirecto de registro, pero insuficientes para los modos relativo, de página, indexado e inmediato. Con anterioridad, habíamos decidido que necesitamos dos palabras para el direccionamiento indirecto y quizás podemos hacer otro tanto para estos cuatro modos.

Observemos, primero, que esta propuesta significa un desaprovechamiento de posibilidades por las muchas redundancias que contiene. Por ejemplo:

Almacenar el contenido de C en A

y Acceder al contenido de C y llevarlo a A

son una misma instrucción, pero se codificarían de manera diferente. Ambas podrían ser sustituidas por una instrucción, llamada MOVE, en la que se especifica la procedencia y el destino de los datos (p. ej. MOVE (C), A).

Por otra parte, muchos de los modos de direccionamiento no tendrían sentido en ciertas situaciones. Por ejemplo, las instrucciones:

Desplazar C índice + 4

y Almacenar D inmediato 327

no tienen significado.

También pueden hacerse algunas simplificaciones. Así, en la práctica no resulta difícil limitar las instrucciones lógicas y aritméticas al registro A. Tampoco es difícil simplificar las instrucciones de incremento y decremento. Con estas últimas, antes decíamos:

"Incrementar la posición de memoria N y dejar una copia del resultado en el registro X", especificando N y X en la instrucción.

Pero la experiencia ha demostrado que basta con limitar las instrucciones de incremento y decremento exclusivamente a registros. Por lo tanto, podemos decir:

"Incrementar el contenido del registro X"

Esta es una instrucción sin dirección y, como tal, no requiere especificación de un modo.

Después de estas simplificaciones, nos queda un total de unas 200 instrucciones válidas. Como sabemos que 8 bits pueden representar un número de 0 a 255, un código de función de 8 bits puede representar todas las instrucciones que deseemos, siempre que prescindamos de una estructura lógica. De esta manera, el formato de instrucción lógica del DIM-1 MK1 se ha convertido

en el conjunto de instrucciones aparentemente arbitrario de un microprocesador.

Nos quedan dos registros que aún no hemos utilizado, y sin duda el lector pensará que se nos han olvidado. Nos referimos a los registros F y SP.

El registro F (registro indicador o de estado) se usa para indicar el resultado de una operación aritmética o lógica. Cada bit representa una condición (o estado) que puede producirse durante la ejecución de una instrucción.

Por ejemplo,

bit 15 indica que el resultado fue cero
14 indica que el resultado no fue cero
13 indica que el resultado fue positivo
12 indica que el resultado fue negativo
11 indica que se ha producido un acarreo positivo ("over flow" o desbordamiento de la capacidad)
10 indica que se ha producido un acarreo negativo ("underflow" o desbordamiento negativo de la capacidad)
9, 8, 7 son señalizadores para acarreos entre las cuatro "rebanadas" de cuatro bits. Estos se usan en aritmética en BCD.

Leyendo el registro de estado después de una instrucción y aplicando una máscara adecuada se mejora la utilidad de las instrucciones de salto condicional.

Finalmente tenemos el registro SP, cuyas siglas significan "Puntero de apila". Cuando escribimos programas, a menudo nos encontramos transfiriendo datos desde un registro a la memoria (porque necesitamos el registro para cualquier otra operación) y desde esta nuevamente al registro, después de ejecutar algunas instrucciones. Estas operaciones constituyen una pérdida de tiempo y espacio porque bloquean posiciones. Para evitar este problema y disponer de un almacenamiento temporal similar, una parte de la memoria se usa a modo de "pila". Normalmente la pila comienza en la parte superior de la memoria y se desplaza

hacia abajo a medida que se van almacenando datos. El puntero de pila contiene la dirección de la siguiente posición disponible en la pila.

La información se coloca en la pila mediante una instrucción "Push" (introducción) y se saca de ella mediante una instrucción "Pull" (extracción), aunque también se emplean otros nombres mnemotécnicos representando a estas instrucciones.

Supongamos que queremos almacenar temporalmente los datos contenidos en A. El procedimiento sería el siguiente:

Programa
↓
Push A (el contenido de A se almacena en la pila)
↓
El programa continúa, y A queda disponible para otra operación
↓
Pull A (el contenido almacenado es devuelto a A)
↓
El programa continúa, usando el contenido original de A

Al llegar a la instrucción "Push", el ordenador obedece lo siguiente:

"Almacenar A en la dirección contenida en el registro SP".

Después se decrementa el contenido del registro SP, para indicar la siguiente posición en la pila (recuérdese que la pila va en descenso desde la parte superior de la memoria).

Al llegar a la instrucción "Pull", el ordenador primero, decrementa el contenido del registro SP, para obtener la última posición ocupada, y, después, ejecuta la orden:

"Acceder a la dirección indicada por el registro SP
y transferir su contenido a A."

Es importante señalar que el manejo de datos en la pila se rige por la norma llamada *LIFO (Last in. First out)*. Su significa-

do es "último en llegar, primero en salir". La pila nos permite manejar subprogramas con facilidad. Tenemos una nueva instrucción llamada

GOSUB N

que dice: "Ir a subprograma que comienza en la posición N"

Emparejada con GOSUB N tenemos la instrucción,

RET (para retorno)

En ella no se especifica dirección habida cuenta de que su significado es:

"Retornar el control a la instrucción a continuación de GOSUB que lo trajo aquí"

Veamos lo que ocurre. Estamos en la isntrucción 007077, y encontramos GOSUB 003220.

Supongamos que el puntero de pila indica que 177775 es la siguiente posición en la pila.

El control de ordenador capta el contenido del contador de instruciones (007077) y lo pone en la posición de la pila indicada por la dirección contenida en SP (177775). SP es ahora decrementado a 177774.

La dirección del subprograma (003220) pasa al contador de instrucciones, ocasionando el salto del programa a la subrutina (subprograma).

Al final del subprograma encontramos la instrucción,

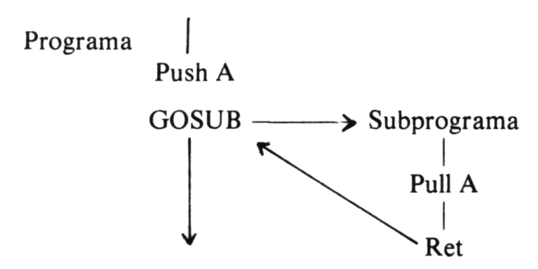
RET

El control del ordenador incrementa el registro SP a 17775. Después accede al contenido de 177775 (que es 003220) y lo lleva al contador del programa. Con esta operación, el contador de

instrucciones contiene, ahora, la dirección desde la que se efectuó el salto. El contador se incrementa en dos (porque GOSUB N es una instrucción de dos posiciones), y ya podemos continuar con la instrucción siguiente a GOSUB.

Aunque el control del ordenador está bastante atareado desplazando posiciones de un lado a otro de la pila, seguramente el lector estará de acuerdo en que GOSUB y RET facilitan la programación de subrutinas. No hace falta decir que se pueden utilizar tantos subprogramas "encajados" como permita el tamaño de la pila hasta que esta, bajando desde la parte superior de la memoria, llegue a la zona ocupada por el programa.

No debemos olvidar que para que la pila y GOSUB sean operativas, las instrucciones PUSH/PULL y GOSUB/RET han de especificarse en pares. Por ejemplo,



no será operativo, ya que la instrucción PULL A lo que hará será acceder a la dirección de la instrucción GOSUB y colocar su contenido (los datos más recientemente almacenados en la pila) en A.

8. LA MAQUINA REAL, Z80

Sin duda, lo tratado hasta aquí ha constituido una dura y larga prueba para el lector, pero al fin ha llegado el momento de poder comparar un microprocesador real con el modelo teórico que hemos diseñado. El dispositivo que estudiaremos es el Z-80 de Mostek/Zilog, de la serie Intel 8080.

Comenzaremos con una breve explicación de la diferencia existente entre un microprocesador y un ordenador. El DIM-1 es un ordenador; y tiene memoria y capacidad para comunicar con dispositivos de entrada/salida. El Z-80, tal como se compra, es una pastilla de 40 patillas (pines, en la jerga) que contiene la U.C.P. (PU en la jerga) de un ordenador. Para convertirla en un ordenador útil es preciso añadirle pastillas de microcircuitos para formar la memoria, entrada, salida, etc. Con la incorporación de dichas pastillas se obtiene un dispositivo útil a la que llamaremos microordenador.

El Z-80 (como la mayoría de los microprocesadores) es un dispositivo de 8 bits. Esto significa que el sistema más conveniente para representar los datos es el hexadecimal. Por ejemplo:

	1001	1110
	9	E
o	1010	0011
	A	3

El Z-80 puede direccionar 64K de memoria, y para ello necesitaremos dos palabras de 8 bits. Por tanto, el Z-80 tiene instrucciones de dirección variable:

a. Instrucciones de una palabra (p. ej.: 81; sumar registro C a registro A)
b. Instrucciones de dos palabras (p. ej.: 06n; cargar registro B inmediato con n)
c. Instrucciones de tres palabras (p. ej.: C3nn; saltar a nn)
d. Instrucciones de cuatro palabras (p. ej.: ED 48 n n; cargar par de registros BC desde la posición n n y nn + 1).

De ello se deduce que una lista de posiciones de memoria, como por ejemplo:

Posición	Contenido
OC50	3A
1	08
2	OF
3	C3
4	FE
5	07

no resulta muy informativa, y por eso los programas suelen escribirse indicando la longitud de la instrucción:

Posición	Contenido	Comentarios
OC50	3A 08 OF	Cargar A a partir de OF80
OC53	3C	Inc A
OC54	FE 07	Comparar Inmediato 07

El control de Z-80 identifica la longitud de una instrucción por su código (tres para 3A, uno para 3C, dos para FE) e incrementa el contador del programa en la misma medida.

Un esquema simplificado del Z-80 se muestra en la figura 18. Como puede apreciarse, las principales conexiones son un bus de dirección de 16 bits (para direccionar 64K de memoria) y un bus de datos bidireccional de 8 bits (para desplazar datos entre la

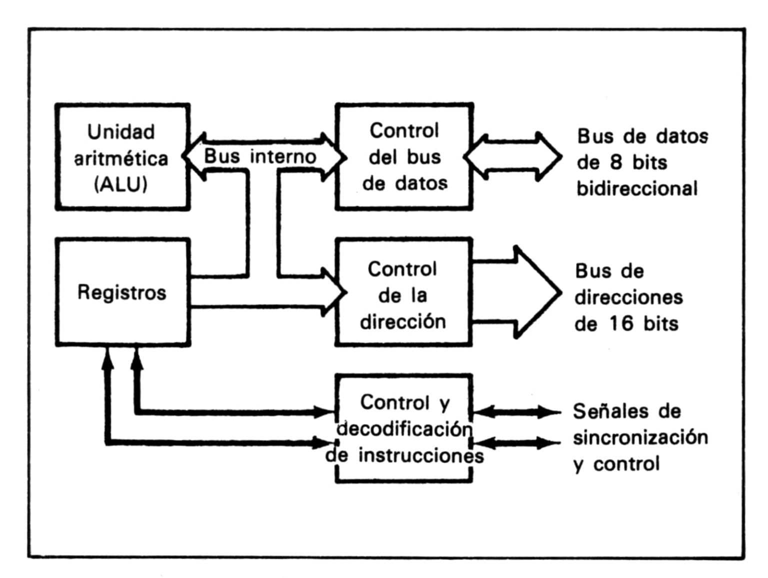


Fig. 18 Diagrama de bloques de la U.C.P. del Z-80.

memoria y los dispositivos de E/S). También hay varias señales de control, tales como las de lectura/escritura, e impulsos selectivos de sincronización ("strobe").

Así, podríamos construir un sistema Z-80 simple como el que muestra la figura 19, que sería el más pequeño posible.

Internamente, el sistema Z-80 tiene 18 registros de 8 bits y 4 de 16 bits, como se muestra en la figura 20. Los registros de 8 bits están distribuidos en dos bloques de ocho registros cada uno, más dos registros para fines especiales (I y R).

A los dos bloques de 8 registros se les llama conjunto de registros principal y conjunto de registros alternativo, y pueden utilizarse indistintamente mediante una instrucción de "intercambio".

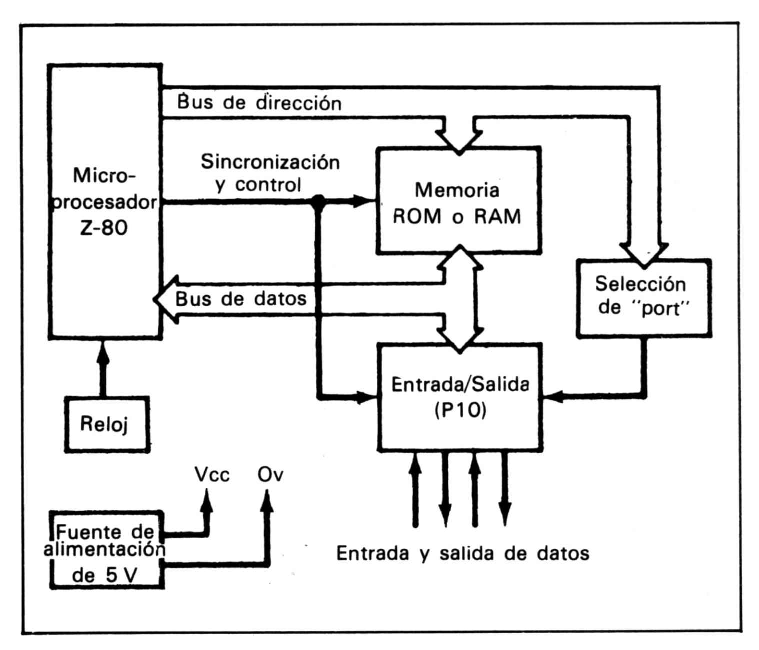


Fig. 19 Sistema Z-80 mínimo.

Cada conjunto contiene 6 registros generales (B, C, D, E, H, L) que pueden ser usados como 6 registros de ocho bits, o como 3 registros de dieciséis bits (formando pares BC, DE, HL).

El registro A de cada conjunto puede utilizarse como registro general o como acumulador para instrucciones lógicas y aritméticas. En el registro HL, y en los dos que se describen a continuación, pueden realizarse funciones aritméticas de 16 bits.

El registro F en cada conjunto es un registro de estado.

Los dos registros restantes son el registro I (vector de interrupción) y el registro R ("refresco" de memoria). El registro I se usa para indicar una dirección indirecta en una interrupción, desde un dispositivo externo. Las interrupciones no se han tratado

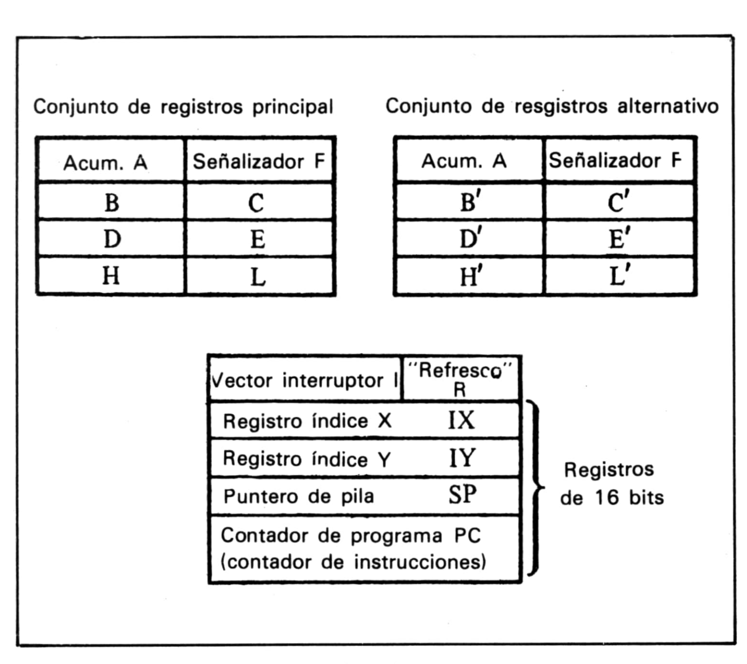


Fig. 20 Conjunto de registros del Z-80.

en este libro por constituir una técnica que se sale un poco de lo que debe ser un manual de introducción. En pocas palabras, puede decirse que la interrupción permite parar la ejecución de un programa, mediante una señal externa, para saltar a otro más urgente.

El registro R se usa para "refrescar" (regenerar o restaurar) memorias dinámicas, y es un registro "invisible" desde el punto de vista del programador.

Los cuatro registros de 16 bits son de tipo convencional; SP es un puntero de pila, PC un contador de programas (de instrucciones) y los registros IX e IY son registros índice. Las funciones de todos ellos son idénticas a las descritas en la especificación del DIM-1.

El sistema Z-80 dispone de los siguientes modos de direccionamiento:

i) Inmediato.
Carga datos de 8 bits en un registro.

ii) Inmediato extendido.
Carga datos de 16 bits en un par de registros.

iii) Relativo.
Como se ha explicado para el DIM-1, este modo consiste en añadir un desplazamiento de 8 bits, con signo, a la dirección en curso.

iv) Dirección extendida.
Es igual al direccionamiento directo en el DIM-1, utilizando dos palabras para especificar la dirección de 16 bits.

v) Dirección Indexada.
Básicamente como se ha descrito para el DIM-1, excepto que se usan dos registros índice para mayor flexibilidad.

vi) Dirección de Registro.
Para operaciones registro a registro (como en el DIM-1).

vii) Indirecto de registro.
El par de registros direccionados contiene la dirección (p. ej., almacenar A en la posición cuya dirección se encuentra en el par de registros HL).

Existen otros dos modos de direccionamiento, llamados de página cero modificada e implícito, que son un poco complejos para un principiante. Como en el caso de las interrupciones, el lector deberá estudiarlos cuando domine los conceptos básicos.

Para resumir las instrucciones del sistema Z-80 nada mejor que el uso de tablas, y las figuras 21 y 22 muestran dos. La primera representa el grupo de carga de 8 bits, y la segunda el grupo aritmético de 8 bits.

Así, el código para: "Cargar registro C con 39" es OE39 (dos palabras), para "Almacenar A en la dirección 01 FE" es 32 FE

01 (una de las sutilezas del Z-80 consiste en colocar primero el octeto de la izquierda), y para "Sumar B y A, resultado en A" es 80.

Como puede verse, en los códigos aritméticos de 8 bits no existe direccionamiento directo.

En total, el sistema Z-80 tiene aproximadamente 160 instrucciones "oficiales" (más algunas "no oficiales" que, presumiblemente, se derivan del funcionamiento de la lógica interna del Z-80) que no es preciso detallar, ya que el lector ahora deberá ser capaz de leer y comprender cualquier manual referente al sistema Z-80 (o al 6800, ó 8085).

FUENTE

DESTINO

		Implícito		Registro			
		I	R	A	B	C	D
Registro	A	ED 57	ED 5F	7F	78	79	7A
	B			47	40	41	42
	C			4F	48	49	4A
	D			57	50	51	52
	E			5F	58	59	5A
	H			67	60	61	62
	L			6F	68	69	6A
Indirecto de registro	(HL)			77	70	71	72
	(BC)			02			
	(DE)			12			
Indexado	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d
Direc. ext.	(nn)			32 n n			
Implícito	I			ED 47			
	R			ED 4F			

Fig. 21 Grupo de carga de 8 bits.

			Indirecto de registro			Indexado		Direc. ext.	Inme diato
E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n
7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
43	44	45	46			DD 46 d	FD 46 d		06 n
4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
53	54	55	56			DD 56 d	FD 56 d		16 n
5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
63	64	65	66			DD 66 d	FD 66 d		26 n
6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
73	74	75							36 n
DD 73 d	DD 74 d	DD 75 d							DD 36 d n
FD 73 d	FD 74 d	FD 75 d							FD 36 d n

	FUENTE			
	DIRECCIONAMIENTO			
	A	B	C	D
'ADD' (suma)	87	80	81	82
'ADC' (suma con acarreo)	8F	88	89	8A
SUB' (resta)	97	90	91	92
'SBC' (resta con acarreo)	9F	98	99	9A
'AND' (función AND)	A7	A0	A1	A2
'XOR' (función OR exclusiva)	AF	A8	A9	AA
'OR' (función OR)	B7	B0	B1	B2
'CP'(comparación)	BF	B8	B9	BA
'INC' (incremento)	3C	04	0C	14
'DEC' (decremento)	3D	05	0D	15

Fig. 22 Aritmética y lógica de 8 bits.

DE REGISTRO			INDIR. REG.	INDEXADO		INMED.
E	H	L	(HL)	(IX+d)	(IY+d)	n
83	84	85	86	DD 86 d	FD 86 d	C6 n
8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
93	94	95	96	DD 96 d	FD 96 d	D6 n
9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
AB	AC	AD	AE	DD AE d	FD AE d	EE n
B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
BB	BC	BD	BE	DD BE d	FD BE d	FE n
1C	24	2C	34	DD 34 d	FD 34 d	
1D	25	2D	35	DD 35 d	FD 35 d	

9. ENTRADA Y SALIDA

Todo sistema de ordenador ha de disponer de algún medio de comunicación con el mundo externo. Generalmente, los fabricantes de microprocesadores proporcionan una pastilla de microcircuito impreso para la entrada y salida de información.

El medio normal de conseguir una función de I/O consiste en utilizar una instrucción de I/O para especificar lo que se conoce como "port" en la jerga. De hecho, este medio es igual a la dirección de dispositivo comentada en la descripción del DIM-1. Así, un teletipo podría ser, digamos, el "port" 3 y un conjunto de indicadores luminosos el 251.

En el Z-80, la instrucción:

D3 n

coloca el contenido del registro A en "port" n, y la instrucción:

DB n

lleva el contenido de "port" n al registro A.

Las comunicaciones pueden realizarse en serie o en paralelo, mediante microcircuitos que descodifican direcciones y proporcionan al dispositivo externo las señales necesarias. A los micro-

circuitos impresos utilizados para la comunicación en paralelo se les llama controladores PIO (*Parallel Input Output,* "entrada/salida en paralelo"), y a los utilizados para la comunicación en serie se les llama UART (Universal Asynchronous Receiver and Transmitter, "transmisor-receptor asíncrono universal"). Pero la transmisión de datos es un tema fascinante que por sí solo puede llenar fácilmente las páginas de un libro.

10. COMO FAMILIARIZARSE CON LOS MICROPROCESADORES

Por muchos libros o artículos que se lean, la única manera de familiarizarse con los microprocesadores consiste en utilizar uno de ellos y aprender de los errores que se van cometiendo. En el mercado actualmente se encuentran muchos kits de evaluación de microprocesadores y ordenadores de uso doméstico ("personales", en la jerga), a precios que oscilan entre las diez mil y las 400.000 ptas., que es interesante conocer y comparar.

Es conveniente clasificar los microprocesadores en ordenadores de entretenimiento, ordenadores de uso doméstico y sistemas utilizados en pequeños negocios. Aunque, en realidad, lo que estamos clasificando es la aplicación, ya que los ordenadores no se ajustan exactamente a estas categorías.

El ordenador de entretenimiento (uno de los hobbies más baratos) es ideal para el entusiasta de la electrónica; consiste en un simple cuadro de mandos y una fuente de alimentación. La entrada se efectúa mediante un teclado hexadecimal, y la salida se obtiene mediante visualizadores ("displays"), asimismo hexadecimales. La programación se hace en código máquina y la pequeña memoria de este ordenador (1K aproximadamente) pronto resulta limitada.

Otros ordenadores más caros utilizados como "hobby" están dotados de un teclado completo (similar al de una máquina de

escribir), y de un interface V.D.U. que permite visualizar los datos de salida en un televisor común. Estos ordenadores suelen, además, estar provistos de salida en cassette, con objeto de que el aficionado pueda crear su biblioteca de programas.

Un punto importante a investigar es la expansibilidad del sistema. Es conveniente (desde el punto de vista financiero) poder adquirir una máquina pequeña que pueda ser ampliada gradualmente a medida que crecen nuestras ambiciones. Obviamente, las ampliaciones que haremos consistirán en aumentar la capacidad de la memoria (por ejemplo, hasta 8 ó 16K), incorporar un intérprete de Basic (para no perder el juicio programando en código máquina), y algún medio de Entrada/Salida como, por ejemplo, una impresora.

Cuando nos ofrezcan el intérprete de Basic, hemos de tener presente que este existe en dos versiones; el llamado Minibasic, que trabaja solamente con números enteros (p. ej. 4057, pero no 4057, 236) y cuyas posibilidades son bastante limitadas, y el Basic 8K, que trabaja con números reales (p. ej. 4057, 236) y tiene muchas funciones de utilidad como pueden ser la raíz cuadrada, la obtención de matrices, etc.

Estos ordenadores para aficionados se venden a menudo desmontados, y el futuro constructor debe saber antes de embarcarse en tal aventura, que el montaje de uno de estos dispositivos tiene un promedio de 2000 puntos de soldadura. A quienes no dominen la técnica de soldadura se les aconseja la elección de dispositivos ya montados, pero si se decide por uno desmontado, será mejor que utilice zócalos para todos los puntos de conexión interna. El costo adicional de esta técnica quedará compensada con creces si alguna vez tiene que localizar un fallo.

Los ordenadores domésticos se venden, normalmente, listos para su uso, y sólo se necesita un enchufe de 13A para hacerlos funcionar. El aspecto de esta máquina resulta más atractivo que el cuadro de mandos del dispositivo de entretenimiento y, por tanto, su presencia en una esquina del salón suele ser mejor tolerada. La mayoría de los ordenadores de uso doméstico llevan

incorporada una V.D.U. (unidad de visualización), y utilizan una grabadora de cassettes común para almacenar los programas.

La mayor parte de los ordenadores domésticos pueden ser ampliados fácilmente para ajustarlos a las necesidades —y al bolsillo— del usuario, y todos ellos tienen Basic, ya sea en la versión Mini o en la de 8K. Sin embargo, como las apariencias también se pagan, un ordenador doméstico es considerablemente más caro que el dispositivo de aficionado de potencia similar.

Finalmente, ocupando el lugar más destacado en el mercado de los microprocesadores, tenemos el ordenador utilizado en pequeños negocios. Esta máquina esta dotada de impresora, V.D.U. y teclado, discos flexibles ("floppy") para almacenar ficheros y de una amplia biblioteca de *software* disponible para el usuario (si este desea comprarlo). Estos sitemas están diseñados para ser *utilizados,* no para educar, y probablemente no enseñarán al principiante lo que él necesita saber.

Si el lector decide comprar uno de estos dispositivos, posiblemente lo más indicada para él sea entrar en contacto con un distribuidor local. El autor de este libro adquirió su ordenador de entretenimiento (con 16K memoria y un Basic de 8K) a un distribuidor local que siempre ha estado dispuesto a prestar servicio postventa, y a intercambiar programas y consejos de manejo. No compre la primera máquina que vea; tómese tiempo y pruebe varias, como lo haría con un aparato de alta fidelidad.

Existen muchas asociaciones y grupos de usuarios de máquinas específicas que intercambian programas, ideas, etc., y que, en su mayoría, publican mensualmente boletines informativos. La pequeña cuota anual de subscripción a una de estas publicaciones es, sin duda, dinero bien invertido.

Si la economía del lector no le permite adquirir una máquina, aún le quedan otras posibilidades; si realiza un trabajo técnico (aunque sólo lo sea remotamente), puede intentar convencer a su jefe para que compre una. El coste tiene una incidencia mínima sobre la mayoría de las empresas, y, en cualquier caso, este tipo de sugerencias suele ser bien vista por el empresario.

La inscripción en algún grupo local del Club de Ordenadores para Aficionados, si los hubiere en su localidad podría ser otra solución. Estos grupos aceptan con agrado la incorporación de nuevos miembros, y no exigen que éstos posean una máquina.

Por último, al lector quizás pudiera interesarle informarse sobre los numerosos cursos de informática impartidos en centros oficiales y privados, ya que algunos son de excelente calidad con relación a las tarifas.

11. CONCLUSION

Con este libro no se ha pretendido hacer del lector un experto en el uso de un determinado microprocesador, sino que se han intentado explicar algunos de los conceptos básicos que suelen omitirse en las descripciones de microprocesadores.

El autor tiene la esperanza de que el lector habrá adquirido los conocimientos suficientes para abordar cualquier manual técnico sobre microprocesadores, con la seguridad de que podrá comprenderlo.

APENDICE 1.

El sistema binario

Es obvio que estamos acostumbrados a contar en unidades, decenas, centenas, etc. Por ejemplo, el número decimal 9317 sabemos que significa, "nueve millares, más tres centenas, más una decena, más siete unidades", y podemos expresarlo de la siguiente manera:

$$
\begin{array}{lr}
9 \times 10 \times 10 \times 10 & = 9000 \\
+ 3 \times 10 \times 10 & = 300 \\
+ 1 \times 10 & = 10 \\
+ 7 & = 7 \\
\hline
 & 9317
\end{array}
$$

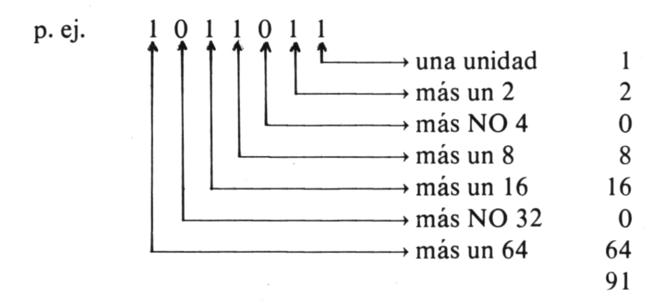
Nuestro sistema de numeración se basa en diez porque tenemos diez dedos, y se llama sistema con base diez. Pero no existe ninguna razón especial que nos obligue a usar diez, puesto que un sistema de numeración puede construirse con cualquier otra base.

Supongamos que tenemos un sistema de numeración con base dos. En tal supuesto, sólo podríamos utilizar dos símbolos, 0

y 1. Las columnas (unidades, decenas, etc. en decimal) serían 1, 2, 4, 8, 16, etc., y contaríamos de la siguiente manera:

Decimal	Binario
0	0
1	1
2	10
3	11
4	100
5	101
6	110 etc.

Para mayor claridad, podemos convertir un número binario nuevamente a decimal:



Es decir, el binario 1011011 equivale a 91 en decimal.

El sistema binario es muy laborioso para el uso cotidiano, debido a que los números se hacen excesivamente largos. Este sistema resulta, sin embargo, ideal para compuertas lógicas con dos estados de salida. El coste de la lógica adicional necesaria para el manejo de número grandes queda ampliamente compensado por la sencillez de los circuitos. El sistema binario se emplea universalmente en todos los ordenadores digitales, aunque el programador no necesita conocer el sistema de numeración interno de la máquina.

APENDICE 2.

GLOSARIO DE TERMINOS DE MICROPROCESADORES

Acumulador

Registro en el que pueden efectuarse operaciones lógicas y aritméticas.

ALGOL

Lenguaje de programación de alto nivel. Es más versátil que el lenguaje BASIC, pero más difícil de aprender.

Almacenamiento

Cualquier dispositivo capaz de recordar. Generalmente tiene el mismo significado que memoria.

Almacenamiento intermedio ("buffer")

Almacenamiento para una palabra en binario, generalmente con una longitud de uno o dos octetos (bytes).

ALU

Siglas de *Arithmetic and Logic Unit* ("unidad lógica y aritmética"). Es la parte de la CPU que realiza las funciones lógicas y aritméticas.

ASCII

Siglas de *American Code for Information Interchange* ("Código americano de normas para el intercambio de la información"). Es un código de 8 bits utilizado para representar símbolos alfanuméricos (letras, números y signos de puntuación). Los datos se representan mediante siete bits, y el bit restante se usa como bit de paridad.

BASIC

Siglas de *Beginners All purpose Symbolic Instruction Code* ("Código de instrucciones simbólicas, para todo tipo de aplicaciones, dirigido a principiantes"). El BASIC es un lenguaje de alto nivel muy fácil de aprender, que normalmente se utiliza como intérprete.

Baudio

Medida de velocidad de un enlace de datos. Un baudio equivale a 1 bit/seg. Algunas velocidades comunes son: 110 baudios (teletipo), 300 baudios (impresora de alta velocidad), 1200 baudios (V.D.U.), 2400 baudios (ordenador a ordenador).

BCD

Siglas de *Binary Coded Decimal* ("decimal codificado en binario"). Cada dígito decimal se representa mediante cuatro bits binarios. Por ejemplo el número decimal 4059 se convertiría en 0100000001011001.

Binario

Sistema de numeración con base dos. Ver Apéndice 1.

Bit

Un dígito binario.

Bus

Vía de transmisión. El ordenador utiliza buses para transferir los datos a las direcciones indicadas (ver figuras 18 y 19).

CAD (ADC en inglés)

Siglas de Conversor Analógico-Digital. Es un dispositivo que convierte una tensión analógica en un número digital que puede ser leído por un ordenador. Estos dispositivos son esenciales para el registro de datos y para el control de procesos.

Cassette

El cassette corriente de la casa Phillips puede emplearse para almacenar programas. La mayor parte de los ordenadores domésticos están dotados de un interface que permite guardar programas en una grabadora normal de cassettes.

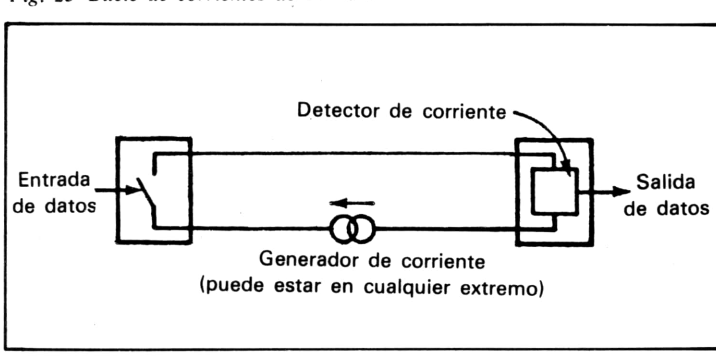
CDA (DAC en inglés)

Conversor Digital-Analógico. (Este dispositivo convierte la salida digital de un ordenador a tensión analógica, de manera que el ordenador puede controlar los dispositivos externos tales como motores, discos indicadores, etc.)

Ciclo 20mA (bucle)

Método común de transmisión en serie. Los datos se envían mediante impulsos eléctricos (ver figura 23) de modo que 20mA representan un '0' y OmA representa un '1'. Dado que se usan corrientes, el sistema tiene excelentes características antirruido.

Fig. 23 Bucle de corrientes de 20 mA.



COBOL

Lenguaje de alto nivel destinado al tratamiento de problemas comerciales. No es adecuado para ordenadores de uso doméstico.

Código máquina

La instrucción que la máquina obedece. Suelen ser muy básicas.

Coma flotante

Básicamente, el ordenador almacena los números como enteros, y esto es un inconveniente cuando se trata de números muy pequeños o muy grandes. Un método más adecuado consiste en utilizar una posición para almacenar los dígitos y otra para almacenar la potencia de diez. Suponiendo que el ordenador trabaja en decimal, podríamos tener lo siguiente:

Número	Posición 1	Posición 2
4057	4057	3
2 356 172	2356	6
1,3792	1379	0
0,00572	5720	–3

Obsérvese que hemos dado por válido que el ordenador trabaja con números entre 0000 y 9999, y por tal razón 2 356 172 se trunca a 2 356 000.

La posición 1 se conoce con el nombre de mantisa y la posición 2 con el de exponente.

La función de coma flotante se realiza por programa, y rara vez es una característica del ordenador.

Compilador

Programa de ordenador para convertir a código máquina un programa escrito en un lenguaje de alto nivel.

Comprobación de errores

La transmisión de datos es susceptible de errores, y la comprobación se hace allí donde pueda producirse alguno a causa de

ruidos. Las técnicas más comunes consisten en una comprobación de suma o en un bit de paridad.

Comprobación de suma

En la transmisión de datos, el ruido puede ocasionar errores, y una comprobación de suma es un medio simple de detectarlos. Los datos se envían en bloques, seguidos de un número binario que indica el número de bits enviados. Por ejemplo:

	Datos	Comprobación de suma
Bloque 1	11010001011011011011011110110001	10011 Correcto
Bloque 2	00000001111011010000000000110110	01010 Error

CORAL 66

Lenguaje de alto nivel. Es poco conocido en España, pero mucho en Gran Bretaña, probablemente porque allí tiene su origen.

CUTS

Término estándar para designar interfaces de cassette.

Datos

Palabra de múltiples interpretaciones. Probablemente la definición más exacta sea la de "información que el ordenador recibe, almacena, manipula y devuelve a su origen".

Depuración

Ningún programa funciona correctamente la primera vez, que se pasa por el ordenador, y la localización de errores en un programa se llama "depuración".

Dinámica

Las memorias de semiconductores pueden ser dinámicas o estáticas. Una memoria dinámica almacena los datos mediante cargas eléctricas en un condensador que han de ser objeto de "refresco" (regeneración) cada milisegundo, aproximadamente. Por otra parte, las memorias estáticas no necesitan "refresco", ya que

éste suele ser proporcionado por la lógica interna del microprocesador.

Dirección

Cada posición de memoria tiene una dirección única mediante la cual las instrucciones del programa pueden acceder a su contenido. La dirección puede ser considerada similar a la etiqueta colocada en una casilla de palomas.

Disco

Los datos de un ordenador pueden almacenarse magnéticamente mediante un método similar al utilizado para grabar música en un magnetofón. Uno de los métodos usados en los ordenadores consiste en un disco (de tamaño similar a un LP) girando a gran velocidad, y varias cabezas de lectura/grabación que se mueven a lo ancho del disco. Los discos se usan como almacenamiento auxiliar.

Disco flexible ("floppy" en inglés)

Medio económico de almacenamiento auxiliar. El disco flexible está hecho de plástico y se asemeja a un disco de 45 r.p.m. Para darle rigidez se usa fuerza centrífuga.

EAROM

Siglas de *"Electrically Alterable Read Only Memory"* ("Memoria sólo de lectura que puede ser alterada eléctricamente"). El usuario puede alterar este tipo de memoria mediante un dispositivo especial llamado programador PROM.

Ensamblador

Programa de ordenador utilizado para convertir a código máquina un programa fuente escrito en código mnemotécnicos.

EPROM

Sinónimo de EAROM. Las siglas significan *Erasable Programmable Read Only Memory* ("Memoria sólo de lectura que puede ser programada y borrada.")

Escritura

Acción de introducir datos en una posición de memoria.

Fichero

Bloque de datos colocados en almacenamientos auxiliares de manera ordenada y accesible (p. ej.: cuentas bancarias, datos de nóminas, etc.).

FIFO

Siglas de *first in. first out* ("el primero en entrar es el primero en salir"). Esta norma se aplica a un tipo de memoria en la que los datos entran igual que lo hace un tren en un túnel, saliendo por el otro extremo en el mismo orden.

Firmware

Término aplicado a los programas almacenados en una ROM.

FORTRAN

Siglas de *FORmula TRANslator* ("traductor de fórmulas"). Es un lenguaje de alto nivel concebido para cálculos científicos.

GIGO

Siglas de *garbage in. garbage out.* En la jerga del sector, esta expresión significa que, por muy brillante que sea un programa, la validez de la información de salida depende de la información de entrada o lo que es lo mismo, la calidad de los resultados es función de la de entrada. En el relato "La máquina que ganó la guerra", de Isaac Asimov, se hacen comentarios interesantes sobre este tema.

Hardware

Las partes físicas de un sistema, por ejemplo, las partes electrónicas, armarios, cables, etc.

Hex (adecimal)

Sistema de numeración con base 16. Es un método muy útil para interpretar números binarios.

Impresora de líneas

Impresora de alta velocidad que, en vez de utilizar una cabeza de tipo móvil, imprime toda una línea de una sola vez.

Impresora por puntos

Impresora de alta velocidad cuya cabeza de tipos consiste en una hilera vertical de patillas (generalmente 7), que son controladas para que golpeen el papel según se mueve la cabeza. Con este método, los caracteres se producen por una combinación de datos (ver figura 24).

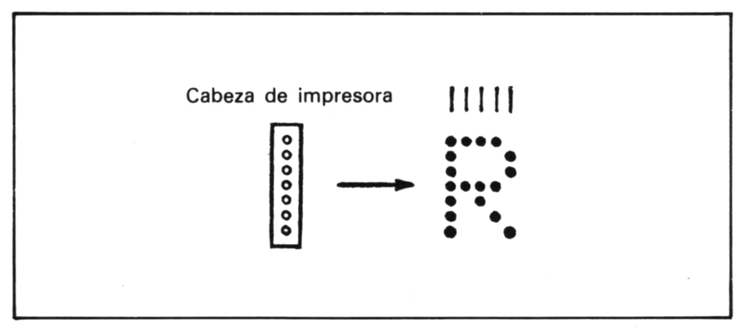


Fig. 24 Impresora por puntos.

Interface

Un interface es la línea "mágica" que separa a una unidad de otra; generalmente se sitúa entre un ordenador y el mundo exterior. Este término suele utilizarse como definición (p. ej.: interface RS232) de niveles lógicos, velocidad de transmisión, etc.

Intérprete

Un lenguaje de alto nivel puede ser utilizado de dos maneras. En la primera, un compilador convierte todo el programa a código máquina, y después el ordenador lo ejecuta prescindiendo del programa de alto nivel. Por el contrario, un intérprete almacena el programa tal como ha sido escrito en lenguaje de alto nivel, y lo va convirtiendo a código máquina a medida que va siendo

necesario. Los intérpretes son lentos e ineficaces, en cuanto al uso de la memoria, pero baratos y fáciles de utilizar. El BASIC suele ejecutarse con un intérprete.

Interrupción

Es conveniente dejar que algún suceso externo interrumpa el desarrollo de un programa para que el ordenador pase a otra tarea más urgente. Estas señales externas se conocen con el nombre de interrupciones.

I/O

Siglas de Entrada/Salida. El término se refiere a la parte de un ordenador que se encarga de las operaciones de entrada/salida y a los dispositivos utilizados en dichas operaciones.

ISO

Código de 8 bits para caracteres alfanuméricos, similar, en principio, al código ASCII, pero diferente en detalles.

K

Abreviatura de Kilo, que significa 1 000. No obstante, en términos de ordenador K representa 1024. Por ejemplo, una memoria de 4K tiene 5 096 posiciones.

Kansas City

Término estándar para designar interfaces de cassette (proviene de una conferencia sobre ordenadores celebrada en Kansas City).

Lenguaje

Un lenguaje de programación es un conjunto de instrucciones definidas de acuerdo a unas normas fijas que puede ser comprendido por un ensamblador, compilador, o intérprete (p. ej.: BASIC, FORTRAN, PASCAL).

Lenguaje de alto nivel

Lenguaje de ordenador que permite escribir programas de manera fácil y comprensible (p. ej.: ALGOL, BASIC).

LIFO

Siglas de *last in. first out* (el último en entrar es el primero en salir). Este término se usa para designar el funcionamiento de una pila de memoria. En el texto principal de este libro se describe detalladamente este procedimiento.

LSI

Siglas de *Large Scale Integration* ("integración a gran escala"). Es la técnica empleada en la Fabricación de pastillas de microcircuitos integrados ("chips").

Memoria

Esta palabra tiene muchos significados. Generalmente se aplica a la capacidad total de almacenamiento de un ordenador, pero también puede designar cualquier dispositivo de almacenamiento.

Memoria auxiliar

Memoria capaz de almacenar grandes cantidades de datos a la que, sin embargo, sólo se puede acceder lentamente (discos, cintas, etc.).

Memoria de núcleos

La mayoría de los microprocesadores tienen memorias de semiconductores, que son baratas pero se destruyen. Pero también pueden construirse memorias que almacenan datos mediante la magnetización de pequeños núcleos. Este tipo de memorias son caras y consumen mucha energía, pero tienen la ventaja de que no se destruyen.

Memoria estática

Memoria en la cual la información está fija en el espacio y en condiciones de ser utilizada en cualquier instante. La memoria estática no necesita "refresco" o regeneración.

Memoria no permanente (volátil)

Memoria que pierde su contenido cuando se desconecta la fuente de alimentación. La mayoría de las memorias de semiconductores son no permanentes (o no remanentes).

Memoria permanente

Memoria cuyo contenido no se destruye cuando se produce un corte en el suministro eléctrico. En las memorias de semiconductores, para evitar que se pierda su contenido, se utilizan baterías auxiliares. Las memorias de núcleos y las ROM son siempre permanentes.

Micro (μ)

Significa 1/1 000 000 ó 10^{-6}. Por ejemplo, 2,2 μF significa 2,2 millonésimas de faradio, y 10 μs indica 10 millonésimas de segundo. También se emplea para indicar "muy pequeño", como en las palabras microordenador, microclima, etc.

Microordenador

Ordenador completo basado en un microprocesador.

Microprocesador

Microcircuito integrado que puede ser utilizado como la unidad central de proceso de un ordenador.

Mili

Significa 1/1 000 ó 10^{-3}. Por ejemplo, 5 ms indica 5 milésimas de segundo.

Miniordenador

Ordenador pequeño, normalmente instalado sobre un pupitre. El límite que separa a un miniordenador de un microordenador no está claramente definido.

Mnemotécnico ("mnemónico" en la jerga)

Literalmente significa ayuda a la memoria (humana). Es un método abreviado de escribir programas en código máquina. Por ejemplo:

"Sumar el contenido de la posición 4057 a A"

podría escribirse como: "Sumar 4057, A".

Nano

Representa 10^{-9}. Por ejemplo, 3 nseg.

Octal

Sistema de numeración con base ocho, conveniente para representar números binarios.

Octeto (byte)

Palabra de 8 bits, que puede representar cualquier número de 0 a 255 (o a – 127 + 127, según desee el usuario). En cierta ocasión fue sugerido el término "rebanada" para nombrar 4 bits, pero tal sugerencia nunca prosperó.

Organigrama

Cuando se escribe un programa, la primera fase consiste en decidir que ha de hacerse, y en qué orden. Para ello se construye un organigrama, detallando los pasos y decisiones que el programa ha de ejecutar. La figura 25 muestra un organigrama típico.

Paridad

En la transmisión da datos, el ruido puede ocasionar errores, y la paridad es simplemente un código de comprobación de errores. La técnica consiste en añadir un bit al total de bits por palabra para obtener un número par o impar (según se use paridad par o impar).

Ejemplo suponiendo que se usa paridad par:

Datos	Paridad	
1011011	0	Correcto
1110010	1	Correcto
1000001	0	← La paridad no coincide

PASCAL

Lenguaje de alto nivel bastante reciente que cuenta con muchos adeptos y al que se ha dado el nombre del famoso filósofo y matemático PASCAL. Es un lenguaje bien ideado que probablemente pronto se hará tan popular como el ALGOL, el FORTRAN y el ampliamente utilizado BASIC.

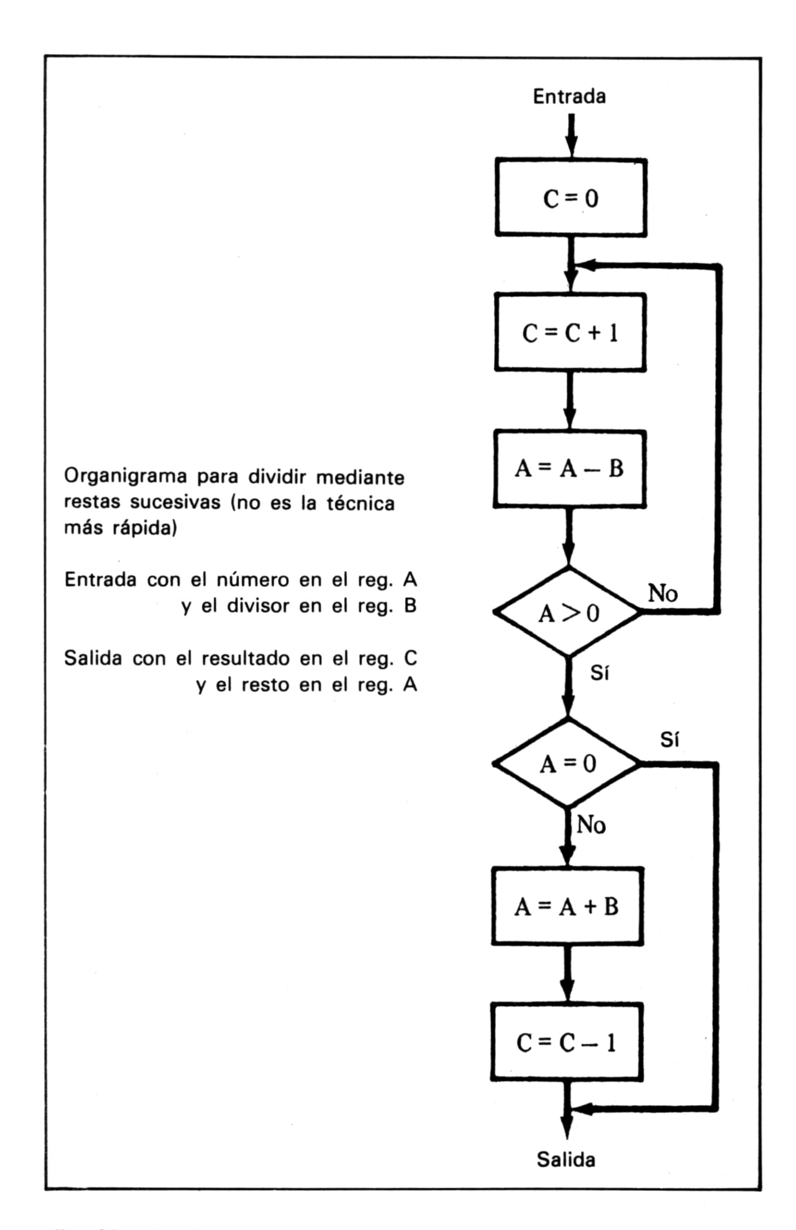


Fig. 25 Ejemplo de organigrama.

Periféricos

Este término se usa para designar los dispositivos externos que comunican con el ordenador, tales como impresoras, lectoras, unidades de representación visual, discos, etc.

Pico

Este término representa 1/1 000 000 000 000, ó 10^{-12}. Por ejemplo, 47p faradios significa 47 billonésimas de faradio.

"Port"

La manera más simple de definir lo que es este término consiste en asociarlo a un conector de salida dotado de una etiqueta específica que permite decir al ordenador: "La salida es el teletipo en 'Port' 3", "La entrada procede de 'port' 27", etc.

Procesador central

Término mediante el que se designa a los ordenadores masivos utilizados en aplicaciones comerciales.

Procesador de textos

Aplicación del ordenador diseñada para ayudar a las mecanógrafas. Permite componer, editar y corregir textos sobre una pantalla, y sacar la versión definitiva a una impresora. Dispone además de memoria auxiliar para almacenar textos estándar (por ejemplo, formularios legales) y utilizarlos siempre que sean necesarios.

Programa Fuente

Programa escrito en código mnemotécnico ensamblador o en un lenguaje de alto nivel, que un ensamblador o un compilador convierte a programa objeto en código máquina.

Programa objeto

Programa en código máquina obtenido a partir de un programa fuente esrito en lenguaje de alto nivel.

PROM

Siglas de *Programmable Read Only Memory* ("memoria solo

de lectura programable"). Es la palabra que suele utilizarse para hacer referencia a una EAROM o EPROM (ver también ROM).

Puertas lógicas

Los elementos básicos internos de un ordenador son las puertas lógicas; circuitos e inversores AND, or, NAND y NOR.

RAM

Siglas de *Random Access Memory* ("memoria de acceso aleatorio"). Este término designa una memoria con las siguientes características:

i) permite el acceso en cualquier orden.
ii) permite grabar y leer desde cualquiera de sus posiciones.

"Refresco" (regeneración)

Las memorias dinámicas almacenan los datos mediante cargas eléctricas en los condensadores. Estas cargas han de ser renovadas aproximadamente cada milisegundo, y a tal proceso se le llama "refresco" o regeneración.

Registro

Extrictamente hablando, un registro es cualquier almacenamiento que pueda contener una palabra, pero generalmente se usa en el sentido de almacenamiento para palabras en la U.C.P. Un programa puede llevar los datos de un registro a la memoria, leerlos de ella y manipularlos.

Reloj

Todas las operaciones en un ordenador están completamente sincronizadas, y el reloj constituye el auténtico "marcapasos" del ordenador. En los diagramas de circuitos, la línea del reloj se representa mediante la letra griega **Φ**.

ROM

Iniciales de *Read Only Memory* ("memoria de solo lectura"). Es una memoria normal (en lo que respecta al ordenador) pero los datos e instrucciones sólo pueden ser leídos de ella. Se usa para contener programas dedicados que permanecen invariables.

Por ejemplo, los intérpretes BASIC de los microordenadores están contenidos generalmente en memorias ROM.

RS232

Se usa en la transmisión de datos a los periféricos y desde éstos, para definir la polaridad de señales y los niveles de tensión.

Salto

Término que designa un salto de instrucciones en la secuencia normal de proceso.

Señalizador ("flag" en inglés)

Un bit de una palabra utilizado para indicar que ha ocurrido algún suceso (p. ej.: señalizador de acarreo).

Software

Básicamente, este término designa las partes de un sistema ordenador que no son visibles, incluyendo el programa, la "filosofía" y la documentación.

Tambor

Otro almacenamiento auxiliar magnético consistente en un tambor giratorio recubierto de un material magnético. Los tambores no son tan populares como los discos.

Teletipo

Impresora para ordenadores, cuyo modelo más común es el Teletipo 33.

Tiempo de acceso

Tiempo necesario para acceder a una posición de memoria y obtener los datos en ella contenidos. El tiempo de acceso es la medida que determina la velocidad de la memoria.

Tiempo de ciclo

Tiempo invertido en acceder a una posición de memoria y leer los datos en ella contenidos. Es la medida de velocidad de todo el ordenador; no sólo de la memoria.

Transmisión en paralelo

El envío de datos de un punto a otro (p. ej.: desde el ordenador a un teletipo) puede hacerse en paralelo o en serie (véase la figura 26).

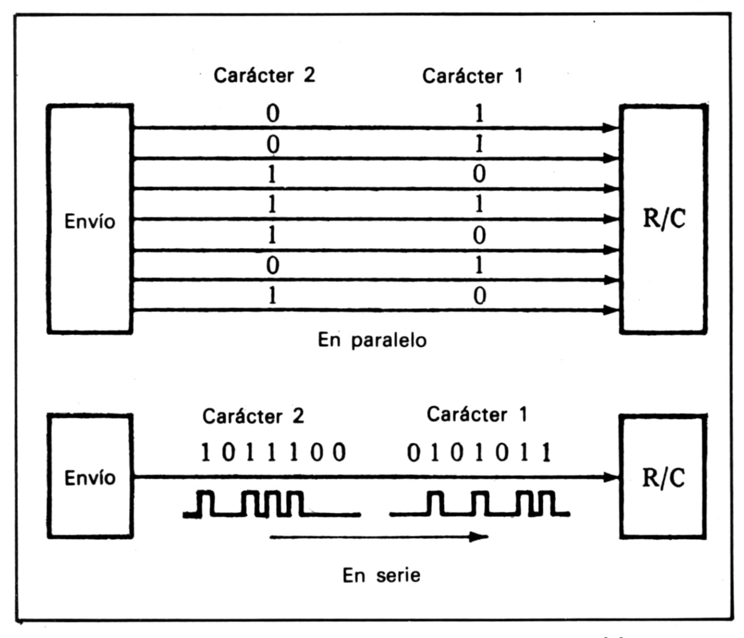


Fig. 26 Comparación de la transmisión en serie y en paralelo.

En la transmisión en paralelo se emplea una línea para cada bit, mientras que en la transmisión en serie se utiliza una sola línea y los datos se envían mediante una serie de impulsos o espacios, o ausencia de ellos.

Obviamente, la transmisión en paralelo en más rápida, pero más cara en términos de cables y circuitos. La transmisión en serie es más común.

Transmisión en serie

Ver transmisión en paralelo.

UART

Siglas de *Universal Asynchronous Receiver Transmiter* ("transmisor-receptor asíncrono universal"). Es un microcircuito impreso para convertir la transmisión en serie a paralelo, o viceversa. Dado que los microordenadores trabajan en paralelo y la mayor parte de los periféricos lo hacen en serie, para la conversión se emplea un UART.

U.C.P. (CPU en inglés)

Siglas de Unidad Central de Proceso. La U.C.P. contiene la ALU del ordenador. De hecho, una pastilla de microprocesador es una U.C.P., aunque el término data de los días en que un ordenador tenía el tamaño de un armario de archivo.

UVPROM

PROM que se borra mediante exposición a la luz ultravioleta, y que puede ser cargada nuevamente con otro programa o con datos.

Vaciado

Un ordenador vacía un programa cuando este ha de ser conservado para volver a utilizarlo. El vaciado puede hacerse en cinta de papel, o en un almacenamiento auxiliar.

V.D.U.

Siglas de *Visual Display Unit* ("unidad de representación visual"). Es un dispositivo periférico que se usa en sustitución de una impresora, y que permite visualizar los datos en una pantalla como la de un televisor. La mayoría de los ordenadores domésticos van equipados con estos dispositivos ya que resulta más barato que una impresora.

Vía de transmisión ("highway" en inglés)

Ordenador formado por bloques (I/O, UCP, memoria) que se comunican mediante vías (véase también bus).

V24

Término estándar para la transmisión de datos, similar a RS232.

Monografías CEAC de Informática

- Iniciación a los Microprocesadores
- Microprocesadores y Computación
- Circuitos básicos de Ordenador

Monografías CEAC de Electrónica

- Proyectos de Control Remoto
- 73 Circuitos Electrónicos Prácticos
- Proyectos con Células Solares

Elementos de Electrónica

1. Componentes y Circuitos Básicos
2. Teoría de la Corriente Alterna
3. Tecnología de los Semiconductores
4. Sistemas y Circuitos de Microprocesadores
5. Telecomunicación